Marcus Eklund

Data Warehousing in the Cloud

—*Analysis of an Implementation Project*

Master's Thesis in Information Systems
Supervisors: Dr. Markku Heikkilä
Dr. Jozsef Mezei
Faculty of Social Sciences, Business and Economics
Åbo Akademi University

Åbo 2019

ABSTRACT

Subject: Information Systems	
Writer: Marcus Eklund	
Title: Data Warehousing in the Cloud - Analysis of an Implementation Project	
Supervisor: Dr. Markku Heikkilä	**Supervisor:** Dr. Jozsef Mezei

Abstract:

Data are generated at an ever-increasing rate. Ideally, all these data should support decision making. One way of facilitating this, is to utilize a data warehouse. Data warehousing is an integral part of business intelligence. The main purpose of a data warehouse is to provide a consistent and non-volatile copy of transaction data from operational sources, in a format suitable for querying and analysis. Implementing a data warehouse through manual labor is time-consuming and difficult. Changes to the data warehouse are slow to implement, and a data warehouse requires constant maintenance. Data warehouse automation is an emerging topic, designed to accelerate and automate development, and ensure quality and consistency through enforced standardization. Through data warehouse automation, changes become non-issues, and expanding and further developing the data warehouse becomes significantly easier.

The author of this thesis works at a company that builds solutions for reporting and analytics, based on data warehousing. The data warehouses are implemented using a DWA-platform developed in-house. The primary objective of this thesis is to research how the established way of working at the company would change, if the on-premises environment were replaced by a cloud-based environment.

The thesis introduces the concept of data warehouse automation and presents some commercial DWA-tools. The theoretical part focuses on the definition and purpose of data warehousing and the components of a data warehouse based on data vault modeling. In addition, cloud computing is discussed, with a focus on data warehousing in the cloud.

To answer the research questions, an artefact was constructed, based on scientific literature and previous project experience. The artefact included tools and services that would replace the ones that are normally utilized by the development team, when working on-premises. The aim was to see whether an artefact could be created that would suit the business case and the established way of working. The artefact was found to be a solid foundation for the design of a real-life implementation. The artefact was implemented in a pilot project for a customer. The project and its results were analysed through case study research.

After finishing the pilot project, it was concluded that the company's way of working is well suited for a cloud-based environment. The offering of the chosen Platform as a service (PaaS) provider included replacements for the tools and services utilized in a standard on-premises business intelligence project. Since the pilot project, several other reporting solutions have been built, utilizing only the cloud components of a PaaS-environment. Cloud-based solutions are now a standard option when designing a solution for customers.

Keywords: data warehousing, data warehouse automation, business intelligence, data vault, Microsoft Azure

Date: 20.4.2018	Number of pages: 114

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ABBREVIATIONS

3NF	Third Normal Form
AAD	Azure Active Directory
AAS	Azure Analysis Services
AD	Active Directory
BI	Business Intelligence
CIF	Corporate Information Factory
DB	Database
DDL	Data Definition Language
DV	Data Vault
DW	Data Warehouse
DWA	Data Warehouse Automation
EDW	Enterprise Data Warehouse
ER	Entity-Relationship
ETL	Extraction, Transformation and Load
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PaaS	Platform-as-a-Service
SaaS	Software-as-a-Service
SSAS	SQL Server Analysis Services
SSDT	SQL Server Data Tools
SSIS	SQL Server Information Services
SSMS	SQL Server Management Studio
SSRS	SQL Server Reporting Services
SQL	Structured Query Language
UML	Unified Modeling Language
XML	Extensible Markup Language

1 INTRODUCTION

1.1 Background

To make informed decisions regarding strategy and management, the management needs to base decisions on facts. The facts can stem from several different sources, but one of the most important sources is the organization's internal data. Transactional sales data, HR-data and similar sources should all be utilized to provide the most accurate and complete data (Turban, Rainer, & Potter, 2007).

To do that, the data must be collected and analyzed, preferably somewhere separate from the transaction systems so the operations are not impacted. Data warehousing solves this problem. However, data warehousing brings on another problem - costs. Designing and implementing a data warehouse and purchasing the necessary hardware can be very expensive, not to mention the time required to complete the project (Turban et al., 2007).

This is partly solved through automation of the implementation. Another solution that is trending now is to migrate towards cloud-based solutions and providers (Grant, 2015).

1.2 Context

The company I work for and write this thesis for has developed an in-house tool called SmartEngine to automate several stages of implementing a data warehouse. The tool imports a Unified Modeling Language (UML) class diagram and turns it into an object model. The tool then proceeds to generate the necessary Structured Query Language code (SQL) to implement the data warehouse and the Extract, Transform, Load (ETL) according to Data Vault Modeling. The tool is presented in depth in chapter 4.1.

The latest addition to the tool is the ability to utilize Microsoft Azure, a cloud-based Platform as a Service (PaaS). By not having to purchase the hardware and maintaining it, the total costs for a data warehouse project are radically lowered. Azure has good capability to scale according to need and usage, and hence the costs will be kept as low as possible. Being a cloud-based solution, low maintenance will be needed on the company's or customer's part.

The company wishes to examine the options available for implementing a fully cloud-based data warehouse. The source data and reports for end-users must also be available online. There should be no reliance on on-premises resources.

A packaged solution has been planned in collaboration with a non-profit service company, owned by cities and municipalities in Finland. The goal is to provide a modular and expandable solution that can be duplicated with relative ease to suit other customers. The end-result and feature list should stay the same between customers, but some work will be necessary to tailor for varying make and usage of source systems.

The business case revolves around modular packages: financial, human resources (HR) and education are separate packages that the customer can purchase one at a time, or all at once. The time from purchase to production is estimated to be 2-3 months per module.

We have recently finished planning a pilot project where we designed a data warehouse completely in the cloud. The customer does not need to invest in any hardware, the only costs are monthly fees according to the scale and performance of the Azure-services. The reports are made in Excel using connections to the semantic layer of the data warehouse, so no new tools or software are needed there either.

1.3 Scope

As the company uses the Data Vault model for implementing a data warehouse, there will be limited reflections to Inmon's or Kimball's methodology (Inmon, Imhoff, & Sousa, 2002; Kimball, Ross, & Thornthwaite, 2011). Both Inmon and Kimball are predominant figures in data warehousing theory, but not having hands-on experience of either of the theories limits the capability to accurately reflect on these methodologies. Both the in-house tool SmartEngine and the commercial product WhereScape model data warehouses using the Data Vault model. Because of that, when data warehousing is mentioned in this thesis, it is assumed to be according to the data vault model unless else is specified.

A second limitation related to tools used, is that the company is a certified Microsoft partner and, as such, it is expected that there is a heavy reliance on Microsoft technologies and tools. Competitors' tools will be discussed, but to a lesser extent.

Because cloud computing is such a broad topic, for this thesis we will look at PaaS providers and applications, as those are used in the company's daily tasks and projects. Cloud security, data governance, and data privacy are such broad topics that they will only be briefly discussed in this thesis.

The company is planning a migration towards Data Vault 2.0. However, all implementations so far have been constructed according to Data Vault 1.0. Because of that, the knowledge of Data Vault 2.0 and its details is fairly limited and, therefore, will not be discussed in this thesis.

The largest part of the research will revolve around data warehouse automation (DWA):

- A definition and introduction to DWA
- The benefits and challenges for both the customer and the supplier
- A brief look at commercial tools for DWA
- A presentation of the in-house tool SmartEngine

Part of the research will revolve around the aspects that need to be accounted for, when moving from an on-premises SQL Server environment to a cloud-based environment, in this case Microsoft Azure.

The company's expectations from the thesis are (among other things):

- That the outcome can be used to prove the statement behind our business cases. Having actual results of projects to present to future and potential customers would be very valuable to the company.

- That the thesis can be used as a source for white papers and blogs to be used in the company's marketing.

1.4 Introduction to DWA

The published research regarding DWA is fairly limited and often has a focus on technical implementation. Consequently, I will present an overview of DWA and DWA tools using a research report by Eckerson Group (Eckerson, 2015b), white papers and blogs, mostly from the providers of such tools. As they are not peer-reviewed, they will be left out of

the literature review, but used in this introduction to provide some information about what DWA and the different tools are capable of today.

> *Data warehouse automation (DWA) tools eliminate the manual effort required to design, deploy and operate a data warehouse. By providing an integrated development environment, DWA tools enable developers and business users to collaborate around designs and iteratively create data warehouses and data marts. This turns data warehouse deployment from a laborious, time consuming exercise into an agile one. (Eckerson, 2015b, p. 2)*

No one can question the importance of a functioning data warehouse (Sen, Ramamurthy, & Sinha, 2012). The data warehouse should contain clean, integrated and reliable data to support decision making. Yet creating a data warehouse is a very time- and effort-consuming project with high risk of failure (Kimpel & Morris, 2013). This is where DWA comes in. Automation has historically been used to improve quality, increase productivity, reduce manual labor, reduce costs and allow for faster delivery (Eckerson, 2015b). The same traits make DWA an attractive and valuable asset for modern data warehouse development.

The purpose of DWA is not remove the human aspect, but rather allow the data analyst to focus on the important aspects of data warehousing.

> *Iversen and other proponents stress that data warehouse automation is not a strategy for replacing human beings. Instead, DWA tools and methods should be used to automate the most repetitive, tedious, or time-consuming aspects of warehouse development, starting with the creation and maintenance of documentation, which is every developer's least-favorite activity. (TDWI, 2015, p. 5)*

DWA is a combination of tools that automate and optimize the process of designing, building, deploying and managing data warehouses. It also forces a change in mindset as data warehouse development takes a more agile approach, through incremental development and being able to react to changes and new data sources. The data warehouse can follow the business needs more closely and adapt at a rapid pace (TDWI, 2015).

DWA allows changes to be made later in the projects, and more frequently without disrupting the development, and without rendering previously done work obsolete. As the automation relies very heavily on design patterns and implementation patterns, generating and regenerating entities quickly and effortlessly results in a fundamental paradigm shift, regardless of the business domain. The design and development process becomes more agile, as changes are easily implemented on the fly. There is no need to feel tied to a certain plan (Rossi, 2015).

1.4.1 Challenges

DWA has not yet reached a solid foothold on the market. Practitioners tend to think that companies do not believe that a DW-project does not have to take years and be incredibly expensive. DW-vendors receive a large part of their profit from supporting and upgrading existing systems and, as such, are not keen on demolishing their business model by suggesting automation. The existing DWA-vendors are fairly small, the largest having a revenue of $50 million. If there existed one large and reputable vendor, the threshold to adopt automation could be lower (Eckerson, 2015b).

The price of a DWA-tool might seem steep. A high upfront payment or expensive licensing might feel alarming, but once the benefits are realized and it is seen how much manual labor is removed, the price does not seem high at all. However, the immediate value is not clear when starting to adopt DWA (Eckerson, 2015b).

The existing DWA-tools are fairly new and focused on traditional relational database systems. Companies are already looking into utilizing big data and big data technologies, such as Hadoop, Spark and others. Once the automation tools catch up with the big data era, the benefits will be even greater. Currently however, DWA-vendors are quite limited in resources and size (Eckerson, 2015b).

Because adoption of DWA can be a daunting endeavor for the aforementioned reasons, care must be taken to address fears and predisposition. Analyzing frameworks such as the Technology Acceptance Model (TAM) and addressing user acceptance is crucial to ensure that DWA is embraced and relied upon. This will be discussed in chapter 2.3.3.

1.4.2 Benefits

The most important benefit of DWA is the removal of manual labor. The tools have different approaches to implementation, but the common theme is to tackle problems during the entire lifecycle. There is a misconception that tools only do small parts such as ETL generation or schema generation, when in fact modern DWA-tools are used in analysis, design, build and generation, transformation, loading and metadata generation. The tools often have options for different modeling techniques and data warehouse schema generation (Eckerson, 2015b; TDWI, 2015).

A key benefit of using DWA is the ability to incorporate new and rapidly changing data sources. Instead of the manual labor of finding all objects the new data relates to, the DWA-tool handles the updating and linking of existing objects automatically according to the data model (TDWI, 2015).

Using DWA results in a standardized product. The tool handles consistency and style. Naming conventions and versioning are handled automatically by the tool. Having standardized environments simplifies development and improves quality. Developers can more easily switch between projects and be familiar with a standard and, thus, shorten the time to start contributing. Standards and optimized code help in minimizing risk (Eckerson, 2015b; TDWI, 2015).

With DWA change management becomes a non-issue. DWA-tools react to changes and modify or recreate objects as necessary to reflect the changes. Any effects further down the chain are automatically dealt with. Logging and documentation are kept up to date. A data warehouse should evolve and grow and using DWA makes this easier (TDWI, 2015).

1.5 Overview of DWA-vendors

1.5.1 Attunity

In 2014, Attunity acquired BIReady, a small DWA-vendor based in the Netherlands. Their product was renamed Attunity Compose. Big investments went into the product to better integrate it with other Attunity products. The product has a very small market share, but as Attunity is a large public company, incorporating the product into their sales pipeline will introduce it to an already existing large customer base (Eckerson, 2015a).

Attunity Compose is a model-driven DWA tool. It can import existing logical models and supports model creation in the tool. The logical model defines business rules, dimensions, hierarchies and key metrics. The tool has built-in intelligence to find data mappings. Based on the model, Attunity Compose can generate the necessary data definition language (DDL) and SQL code to create the tables and execute all data loading and transformation, according to Third Normal Form (3NF) or Data Vault modeling. The tool removes the need for a separate ETL tool and runs all the loading and transformation jobs, monitors and logs activity, and generates data marts and dimensions. Attunity Compose supports most major platforms, such as SQL Server, Oracle, Exadata, Teradata and others (Eckerson, 2015a).

1.5.2 Kalido

Kalido started out as an internal company for Shell Oil Company in 2003. Its purpose was to provide data services to Shell Business units. In 2009 Kalido debuted to the public and currently has about 150 customers. In 2014, the company was acquired by Silverback Ventures and transformed into Magnitude Software, with the product still called Kalido (Eckerson, 2015c).

According to Eckerson Group (Eckerson, 2015c), Kalido is the oldest and the most mature data warehouse automation tool. It uses a purely model-driven approach for schema generation and data transformation. The strength of this comes from a visually and functionally impressive modeling engine, allowing business users and developers to collaborate. Much of the technical detail is hidden to make the design logical and business oriented. Using metadata generated by the modeling engine, the data warehouse engine can modify, revisit and rebuild the objects in the staging area and data warehouse. Very little ETL-work is needed. One added benefit of Kalido is the generation of business metadata for BI-tools, such as SAP Business Objects, Cognos, Qlikview, SSAS and Excel. Kalido supports SQL Server, Oracle and Teradata (Eckerson, 2015c).

1.5.3 TimeXtender

TimeXtender was founded in 2006 by a Danish BI-consultant. His motivation was that there must be a better way to build data warehouses than by manual labor. TimeXtender has 40 employees in two separate locations, Denmark and Redmond, Washington. TimeXtender has focused on Microsoft technologies and leveraged the Microsoft partner

network and now has more than 2600 customers in 60 countries, generating more than $10 million dollars in revenue (Eckerson, 2015d).

TimeXtender has created a niche for itself, by only committing to Microsoft products and technologies. The product TX2014 only runs on a Microsoft SQL Server and generates data warehouses using Microsoft SQL Server tools. As TimeXtender sells primarily through Microsoft partners and resellers combined with offering affordable prices, the company has reached a substantial customer base in a short time, becoming a DWA-vendor with almost ten times the customers compared to other vendors (Eckerson, 2015d).

Committing itself to Microsoft SQL Server has allowed TimeXtender to offer deeper integration with SQL Server. TimeXtender can automatically migrate code from one version of SQL Server to another. This makes upgrading substantially easier. TimeXtender utilizes SQL Server Information Services (SSIS) for extraction and moving of data. All the transformation code is native SQL code, generated by the software. TimeXtender supports both a model-driven and a data-driven approach. Models can be built in the software, or existing databases or data warehouses can be reverse engineered. Another feature is the ability to connect to existing SQL Server data and start utilizing automation without having to rebuild everything (Eckerson, 2015d).

TimeXtender supports all SQL Server versions from 2008 onward. It can generate SSAS data cubes from the software and this is the primary interface for end-users. TimeXtender also has a plugin for Qlikview to create Qlikview models directly from the SQL Server data warehouse (Eckerson, 2015d).

1.5.4 WhereScape

WhereScape was founded in 2001. Wells Fargo acquired the company in 2003 and that allowed WhereScape to publicly launch its tool. The company started out in New Zealand and still has its headquarters there. Today, more than half of the company's revenue comes from the USA (Eckerson, 2015e).

WhereScape has 700 customers around the world and in different industries, with a majority residing in the USA. The primary use cases and reasons to purchase WhereScape are creating subject-area data marts and creating new data warehouses. Added benefits

include prototyping, supporting data migration and replacing hand-written ETL. There is no limit to the number of data warehouses or data marts, the licensing is done on a named user basis (Eckerson, 2015e).

WhereScape offers two products. WhereScape RED is the development tool for building, deploying and managing data warehouses and data marts. It is completely data-driven and does not contain any modeling. The existing data are imported and used to start building the physical model. RED is capable of building various models, including third normal form (3NF), snowflake scheme, star scheme and data vaults. What sets RED apart from others is that it operates directly on the source data, by means of a metadata repository in the target data warehouse. All native SQL and stored procedures for creation of the data warehouse structures are generated automatically. WhereScape has a built-in scheduler to coordinate ETL jobs, controls versioning using a metadata repository, and produces technical and business documentation (Eckerson, 2015e).

The second product is WhereScape 3D that allows the customers to examine and analyze source data and test target schemas and prototypes with actual data. 3D can also be used to test external data models, or the models created in RED (Eckerson, 2015e).

RED runs on Windows desktops and uses the target data warehouse as the repository for metadata and jobs. RED supports numerous target databases and is considering utilizing cloud-based solutions, such as Amazon Redshift and Microsoft Azure. RED is also used to create OLAP cubes, mainly targeting Microsoft SQL Server Analysis Services (Eckerson, 2015e).

1.6 Research questions

Is the company's established way of working still suitable, when moving towards a cloud-based solution?

Cloud computing is interesting in many ways, when considering Business Intelligence. The prospect of not having internal servers and infrastructure that need maintenance and regular upgrades is tempting. Avoiding up-front investments, and enabling activation of a service for trials and shutting it down when the trials have finished, offer a flexibility and low-risk prototyping that on-premises solutions cannot compete with. However,

cloud computing comes with its own limitations and drawbacks. How do the benefits and disadvantages weigh up?

How does SmartEngine compare to commercial DWA-tools?

When development on SmartEngine began, there were no suitable commercial tools available. SmartEngine relies on a slightly different approach than other DWA-tools, as the tool is a model implementation platform. No modifications or work is done in the tool itself. The capabilities of the tool can be expanded by discovering the patterns and rules for implementing a certain type of model and then building the application logic for it.

2 LITERATURE REVIEW

2.1 Data warehousing

2.1.1 Definition

Business Intelligence (BI) is not a new concept. One early occurrence of the term is by H.P. Luhn (1958), who defined BI and introduced a business intelligence system as early as 1958. The need for automation and new technologies were driven much by the same reasons as now:

> *It has become evident in recent years that present communication methods are totally inadequate for future requirements. Information is now being generated and utilized at an ever-increasing rate… (Luhn, 1958, p. 314)*
>
> *At the same time the growth of organizations and increased specialization and divisionalization have created new barriers to the flow of information. (Luhn, 1958, p. 314)*

Luhn continues to describe in detail what can be called an early decision support system. The technology available was, according to Luhn, not at an adequately advanced level to create the system defined in the paper (Luhn, 1958). Thirty years later, Devlin & Murphy (1988) present the EBIS-architecture (E|ME|A Business Information System) employed by IBM. Devlin & Murphy (1988) proceed to demonstrate that querying transaction data and requesting information from the IS organization when needed is not a viable long-term solution. Similar issues are presented by several others (Golfarelli & Rizzi, 2009; Linstedt & Olschimke, 2015). Devlin and Murphy introduced the term Business Data Warehouse as *"The BDW is the single logical storehouse of all the information used to report on the business.". (Devlin & Murphy, 1988, p. 67)*

One of the first articles and one of the most important papers mentioning a relational model of database systems was written by Edgar F. Codd in the 1970s (Krishnan, 2013). The paper was pivotal for several reasons. One was that it was the first to introduce the idea of managing data using a relationship-based approach. Another reason was introducing the idea of abstracting the management and storage of data from the user. It

further discussed the idea of removing duplicates and minimizing redundancy (Krishnan, 2013).

According to Inmon (2005), a data warehouse is a subject-oriented, integrated, time varying, non-volatile collection of data in support of the management's decision-making process. Kimball's definition of a data warehouse is that it is a copy of transaction data specifically structured for query and analysis. (Kimball et al., 2011). Some distinguishing characteristics of a data warehouse as defined by Turban, Rainer & Potter (2007) and Inmon, Imhoff & Sousa (2002) are:

- **Organized by subject.** The data are organized by subject, e.g. by customer, vendor, product, category. The grouping supports decision support and data analysis.

- **Consistent**. All data must be coded in a consistent manner. Before loading the data, they must be transformed so that all units and data are uniform.

- **Historical**. Data are never removed from a data warehouse, and the historical data can be used for analyzing trends and for forecasting.

- **Non-volatile.** Once data have been loaded into the warehouse, no updates or changes on the data will be made.

- **Uses online analytical processing (OLAP).** Typical operational systems rely on online transaction processing (OLTP). For a point of sale system, it is imperative that all transactions are processed as soon as they occur. There is rarely a need to look back at historical sales in an OLTP-system. Therefore, the focus is on speed and efficiency. The emphasis is on data entry and data modification speed, not analytical or querying capabilities. A data warehouse is not a mission-critical system and, instead, processes a large batch of data separate from the transaction systems not to slow down daily operations (Conn, 2005).

- **Multidimensional.** A normal relational database stores data in two dimensions. A data warehouse is designed for more than two dimensions. The purpose is to be able to analyze for example sales data and group it by vendor, by time period, or

by geographic area among other attributes. The multidimensional representation of data is most often a data cube.

- **Relationship with relational databases.** The data from operational OLTP systems are loaded into the data warehouse through a process called extract, transform and load (ETL). Gathering the data from operational systems means that the OLTP-systems are not impacted during the actual analysis. It also means that the data will be current and there is no need for additional manual labor to enter the data into the data warehouse.

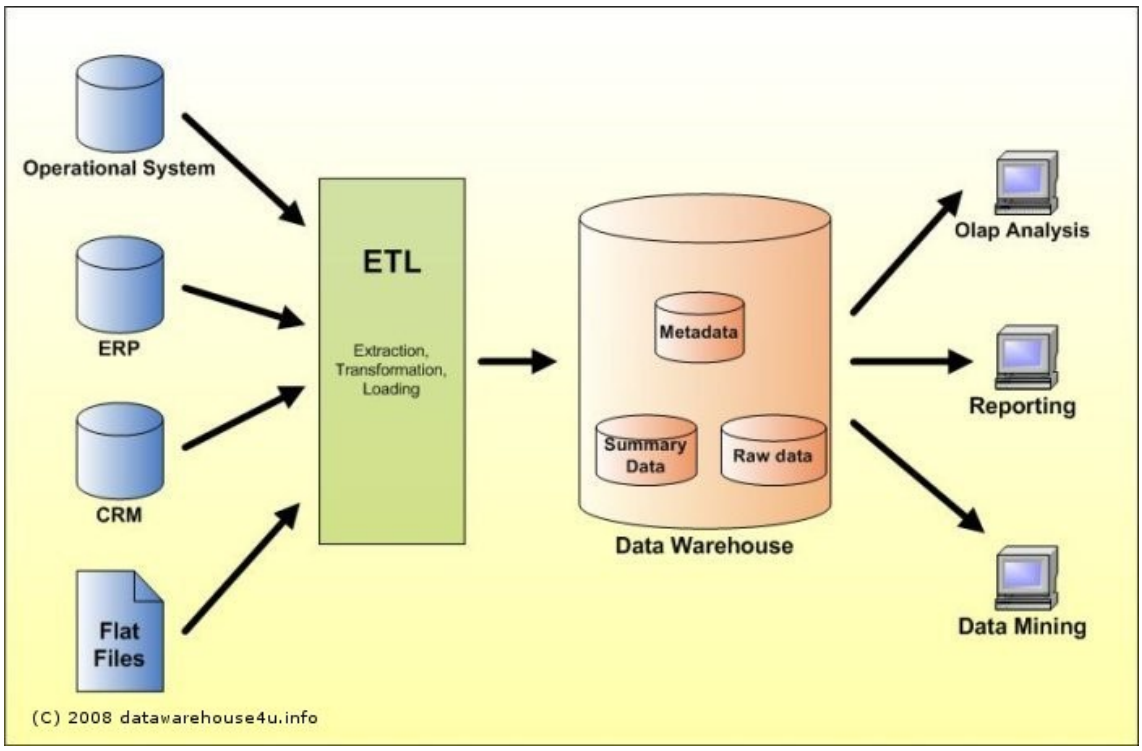


Figure 1 - Data Warehouse Architecture (http://datawarehouse4u.info/images/data_warehouse_architecture.jpg)

As seen in Figure 1, a data warehouse system usually consists of several major parts, but is not limited to:

- **The source data**, which usually come from operational systems.
- **The staging area (ETL)**, which manages the extraction of data from source systems, transforms and cleans up the data, and then manages the inserting of the data into the DW.
- **The data warehouse databases**, where the actual data are stored, along with metadata.

- **Presentation layer,** either using data marts or directly providing the data for further use and analysis.

Kimball & Ross (2013) describe several goals for a data warehouse. The data warehouse must make information accessible, maintain consistency of information, be able to adapt to change and serve as a definitive foundation for improved decision making.

They further state that it is vital that the business accepts the data warehouse. They must see the data warehouse as the best available source of information, otherwise it might not even be used at all. This claim is supported by case studies carried out by Watson, Goodhue & Wixom (2002).

There are two different main methodologies for choosing a data warehouse architecture. There is the top-down approach of modeling a Corporate Information Factory, in third normal form by Bill Inmon (Ariyachandra & Watson, 2006; Breslin, 2004; George, 2012), and the bottom-up approach of dimensional modeling by Ralph Kimball (Ariyachandra & Watson, 2006; Breslin, 2004; George, 2012), known as the data mart bus architecture. In their survey-based study, Ariyachandra & Watson (2006) found that:

> *The bus, hub-and-spoke, and centralized architectures earned similar scores on the success metrics. This finding helps explain why these competing architectures have survived over time—they are equally successful for their intended purposes. (Ariyachandra & Watson, 2006, p. 8)*

	Inmon	Kimball
Building Data warehouse	Time Consuming	Takes lesser time
Maintenance	Easy	Difficult, often redundant and subject to revisions
Cost	High initial cost; Subsequent project development costs will be much lower	Low initial cost; Each subsequent phase will cost almost the same
Time	Longer start-up time	Shorter time for initial set-up
Skill Requirement	Specialist team	Generalist team
Data Integration requirements	Enterprise-wide	Individual business areas

Figure 2 - Pros and cons of both the approaches (http://www.computerweekly.com/tip/Inmon-vs-Kimball-Which-approach-is-suitable-for-your-data-warehouse)

As seen in Figure 2, building a data warehouse according to Inmon's design is quite time-consuming. The initial effort is much larger than with Kimball's approach. However, once the initial development is done, further development is considerably faster and easier. Maintenance and upkeep require less effort compared to dimensional modeling (George, 2012).

2.1.2 Dimensional modeling

Ralph Kimball is a proponent for dimensional modeling, stating that dimensional modeling is the best way to present information as simply as possible (Mundy & Thornthwaite, 2007). The modeling will follow the business process and, as such, will be able to quickly and accurately return query results and information to users.

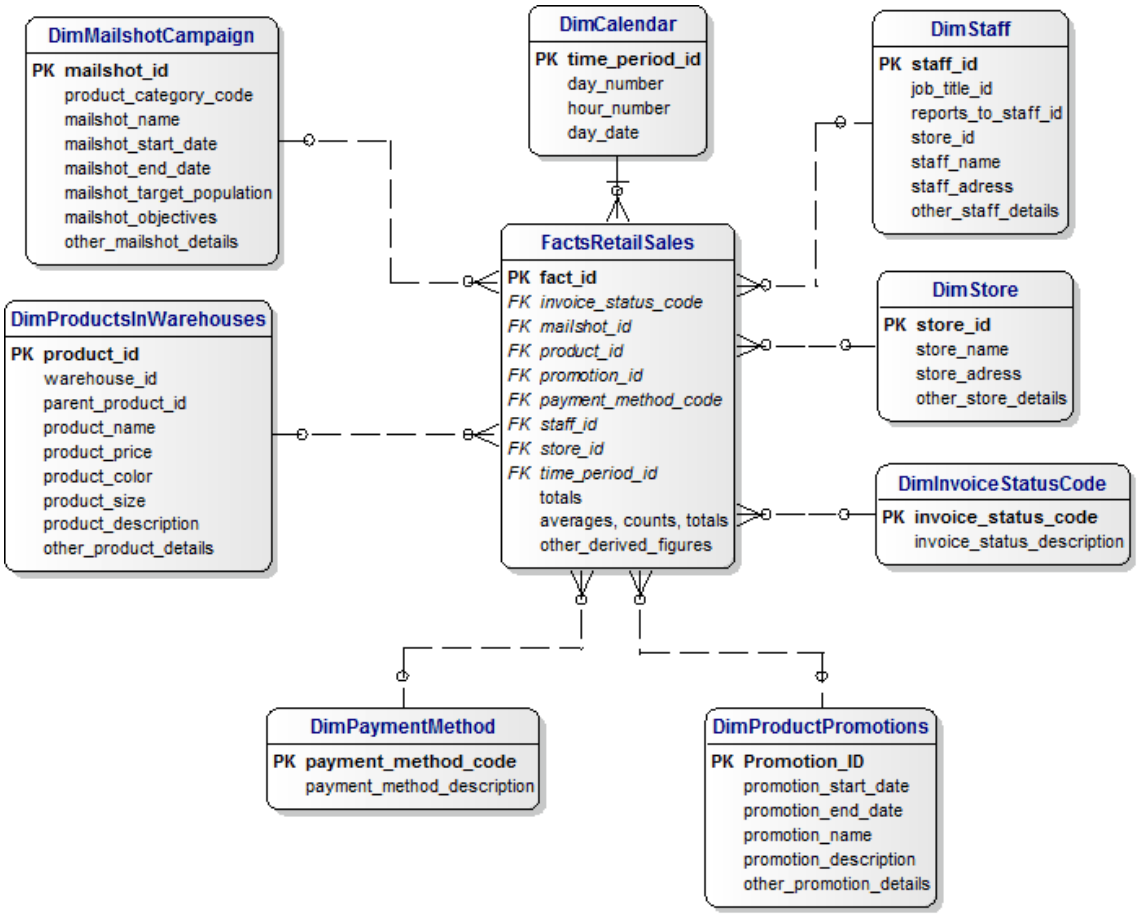


Figure 3 - An example of Dimensional Modeling (http://www.databaseanswers.org/data_models/retail_customers/pragmatic_dimensional_model.htm)

Dimensional modeling involves using facts and conformed dimensions to describe the data, as seen in figure 3. The facts typically contain numerical values and measurements, while the dimensions are descriptive, used for grouping and labelling. Conformed dimensions are meant to be used for all facts combined and, as such, they are normally designed first. The benefit of dimensional modeling is a fast and objective-focused modeling approach that provides quick results and is quick to implement (Kimball & Ross, 2013).

The dimensional modeling is often referred to as the bottom-up approach. Small and specific data marts are designed to suit a certain need. Data marts are implemented next to each other and then combined to become the data warehouse (Jukic, 2006; Kimball & Ross, 2013).

2.1.3 Corporate Information Factory

Breslin (2004) defines the Corporate Information Factory (CIF) as:

> *Inmon's architected environment consists of all information systems and their databases throughout a given organization. He calls this behemoth the Corporate Information Factory, or CIF (Inmon and Imhoff, 2002). (Breslin, 2004, p. 8)*

Instead of just considering a data warehouse alone, the entire IT-ecosystem is seen as a whole. This leads to an evolutionary lifecycle, where the data warehouse follows the design methods and technologies used to develop the operational systems (Breslin, 2004).

The data warehouse in a CIF was presented by Bill Inmon as an integrated database modeled using the traditional Entity-Relationship (ER) modeling, normalized to the third normal form (Inmon et al., 2002). The data warehouse will contain the most detailed version of data at the lowest granularity. Dimensionally modeled data marts are then designed and implemented with the data warehouse as source (Jukic, 2006).

2.1.4 Future of data warehousing

Data warehousing is one of the most important projects in the IT-industry. Data warehousing has experienced large growth since 2000 and continues to grow. Decision

support systems have evolved into more robust and advanced applications (Kimpel & Morris, 2013; Sen et al., 2012).

However, successfully implementing and finishing a data warehousing project is no easy task. An estimated 40% to 50% of data warehouse initiatives fail, most with costly consequences (Kimpel & Morris, 2013; Sen et al., 2012). Kimpel & Morris (2013) report on the success factors found through a survey answered by 98 data warehousing professionals:

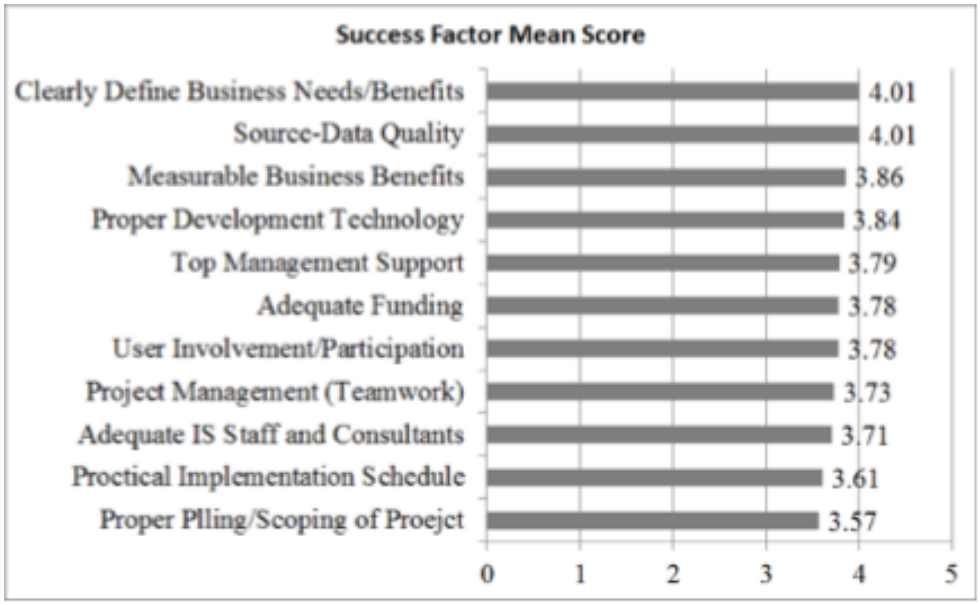


Figure 4 - CSF Mean Score (Kimpel & Morris, 2013, p. 379)

One way to avoid failures and improve results is to focus more on the maturity of the data warehousing processes. Correct technological solutions and decisions are not among the top critical success factors, instead understanding the needs of the users, and being able to measure the business value provided by the data warehouse are key factors.

> *It is therefore critical for the DW community to devote more thought to understanding what afflicts DW design, development, implementation, and management. DW initiatives often end up as failures because of factors such as slipped schedules, unacceptable performance, expandability problems, poor availability, complicated tools, poor data quality, and unhappy users [2], [31]. (Sen et al., 2012, p. 336)*

Possibly the biggest and most important research topic revolves around developing and adopting data warehousing and OLAP methodologies to support analytics over big data (Cuzzocrea, Bellatreche, & Song, 2013; Šubić, Poščić, & Jakšić, 2015).

The sheer volume and complexity of big data causes major issues when trying to implement practical applications (Assunção, Calheiros, Bianchi, Netto, & Buyya, 2015). The potential size of unstructured data, the explosive number of dimensions, the technical challenges of designing an OLAP data cube for big data, and sufficient end-user performance are open and important research problems (Cuzzocrea et al., 2013). To solve this, several new models are being developed, such as Data Vault, MapReduce, Hadoop and Hive (Krishnan, 2013; Šubić et al., 2015).

> *...heterogeneity is permanent in the future of the data warehouse, and the concept remains the same as defined 40 years ago by Bill Inmon, but the physical implementation will be different from the prior generations of data warehousing. (Krishnan, 2013, p. 217)*
>
> *We mentioned the Data Vault and Anchor modelling methods that are designed just for the purpose of storing and processing large amounts of data in data warehouses. These models may have come too late to the market, because most of the big companies like Google, Amazon and Facebook have already switched to non relational solutions. (Šubić et al., 2015, p. 242)*

Some future research directions suggested by Cuzzocrea, Bellatreche, & Song (2013, p. 69) are:

- Methodologies for designing OLAP data cubes over big data
- Innovative solutions for computing aggregations
- High-performance architectures for implementing OLAP data cubes over big data, e.g. utilizing GPU processors.
- Optimization to MDX-language
- Semantically-rich OLAP big data cubes

Krishnan (2013) presents data virtualization as an emerging technology for next-generation data warehouses. Data virtualization would be used to create a semantic data integration architecture. The aim with data virtualization is to create a single presentation layer to the consumer. This is done through semantic transformations and by abstracting

the infrastructure. Instead of physical transformations, data are integrated through virtualizations. As the data are not moved, the database or data store can be optimized for data access, thus fully utilizing the underlying infrastructure. Data virtualization enables automated data discovery, adding new data and data sources. Contextualization can be done during discovery to reduce errors and speed up the integration (Krishnan, 2013).

Another current data warehouse research topic is real-time data warehousing coupled with real-time analytics (Kakish & Kraft, 2012). In traditional ETL-processes, the most current information is not always available. Kakish & Kraft (2012) state that the current periodic refreshes must give way to continuous updates. Instead of relying on batch runs and offline updating, one possibility for real-time ETL would be to utilize data streams of events from data stores to the data warehouse. However, this is an expensive and difficult architecture and requires further research. As they point out:

> *The importance, complexity and criticality of such an environment make real-time BI and DW a significant topic of research and practice…* (Kakish & Kraft, 2012, p. 8)

As the data sets grow larger and the need for processing power increases, new technologies will be looked at to alleviate the problem. Hadoop is an example of a distributed file system and framework for handling and analyzing large data sets (Shvachko, Kuang, Radia, & Chansler, 2010). Hadoop partitions and distributes the data and computation across vast numbers of hosts, which enables parallel processing. Application data and metadata are stored separately. Metadata are stored in NameNodes. Application data are stored in DataNodes. Application data are replicated for reliability and increased bandwidth (Shvachko et al., 2010). By default, every block is replicated at three separate nodes. The NameNodes keep track of where the blocks are, and handle the replication and writing of new data. If one of the replicating nodes does not answer within a certain time limit, then the block is replicated elsewhere and the node is considered lost. This ensures that the data are protected, without the need for local redundancy or RAID-type technologies (Shvachko et al., 2010).

> *Hadoop clusters at Yahoo! span 25 000 servers, and store 25 petabytes of application data, with the largest cluster being 3500 servers. One hundred other organizations worldwide report using Hadoop. (Shvachko et al., 2010, p. 1)*

Since 2010, Yahoo has increased its usage to 40 000 computers, reaching 455 PB of total storage, and having 4 500 nodes in the largest cluster (Miah, Hasan, & Uddin, 2015).

Hive is a petabyte scale data warehouse, built upon Hadoop and developed by Facebook (Thusoo et al., 2010). As Hadoop, Hive is also an open source project. The aim is to use Hadoop and support queries in a language that resembles SQL, called HiveQL. Instead of traditional RDBMS, Hive utilizes the distributed file system of Hadoop (Thusoo et al., 2010). Scalability is ensured by being able to add new nodes to the cluster, or upgrade existing ones (Shvachko et al., 2010; Thusoo et al., 2010). As of 2015, Facebook´s Hive data warehouse held 300 PB of data in 800 000 tables, or 900 PB considering the 3-way replication. Four PB new data are generated daily (Bronson, Lento, & Wiener, 2015).

2.2 Data Vault

There is very little reviewed and published material on the topic of Data Vaults. The creator Daniel Linstedt has written several books and white papers, but they have surprisingly few citations. It might be that the topic is too new and the model is not yet in extensive use. The following chapters will rely heavily on one of Linstedt's newest books, *Building a Scalable Data Warehouse with Data Vault 2.0*.

Data Vault modeling was invented by Daniel Linstedt in the beginning of the 1990s and released for review in 2002 (Linstedt & Olschimke, 2015). Linstedt was employed by the Department of Defense to design and build a scalable data warehouse. This was before the time of Big Data. There was a clear need for an architecture capable of accommodating large data sets, which were to be processed and loaded quickly and reliably. Scalability and iterative development were also high priorities for the project (Linstedt & Olschimke, 2015).

2.2.1 Architecture

The Data Vault architecture follows the commonly found three-layer architecture introduced by Inmon (2005). However, to suit the needs of an enterprise data warehouse (EDW), certain modifications must be made. Some of the modifications to the typical three-layer architecture are (Linstedt & Olschimke, 2015):

- The staging area does not store historical information and does not apply changes to the data other than ensuring the correct data type.
- The data warehouse layer is modeled according to the Data Vault modeling technique.
- Information marts that are dependent on the data warehouse layer
- Several optional vaults in the data warehouse layer, such as Metrics Vault, Business Vault and Operational Vault. These are used for optimization and for metadata purposes.

2.2.2 Components

The core components of the Data Vault modeling technique are called hubs, links and satellites. They are based on entities from a hub-and-spoke architecture. Hubs, links and satellites from a complex network, similar to a human brain which is a network of neurons (Linstedt & Olschimke, 2015; Williams, 2015).

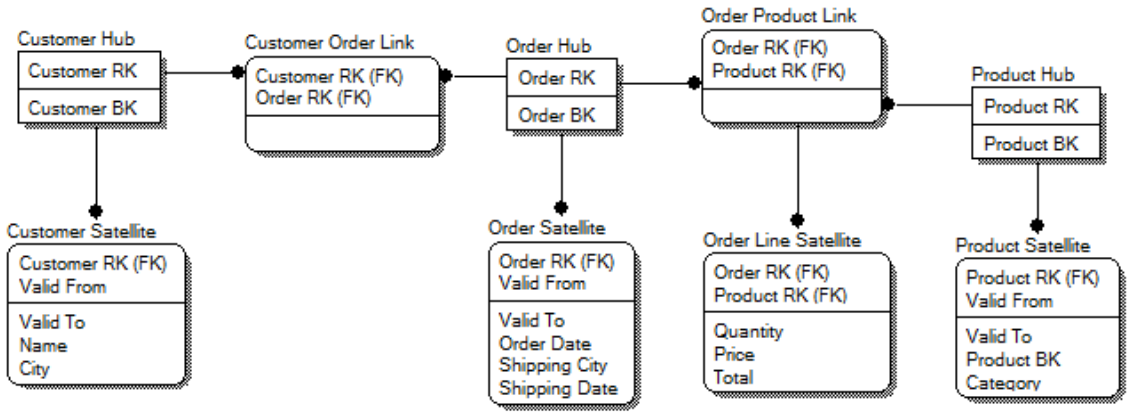


Figure 5 - A Data Vault Model (http://bukhantsov.org/2012/04/what-is-data-vault/)

Figure 5 demonstrates the hubs, links and satellites of an exemplary data vault model when modeling the entities customer, order, and product. Surrogate keys are used in the data vault structure to increase lookup performance. Business keys can sometimes be long combinations of several values, and using a surrogate key in form of a simple increasing integer will result in much faster joins. The surrogate keys of links and satellites are always foreign keys, pointing to the surrogate key of a hub (Linstedt & Olschimke, 2015).

The loading of the tables is presented in chapter 4.1.4.

2.2.2.1 Hubs

Business keys are used to identify certain business objects in operation systems. A business key might be a customer number, an invoice number or something similar, that

makes the business object uniquely identifiable. It might also be a combination of values, such as a customer number and a country code.

Hubs are lists of unique business keys which contains metadata about when the business key arrived for the first time in the data warehouse and from what source system it originates. The primary key of a hub is the surrogate key of its business key. Often, though not mandatory, the hub also keeps record of when the business key was last seen in a loading process (Linstedt & Olschimke, 2015).

2.2.2.2 Links

Links are responsible for modeling associations and transactions between business objects. A link connects related business keys and in data vault modeling, links are modeled between hubs. Having links allows changes to the structure over time. A primary key for the link will be formed by combining the two business keys being linked. This will be stored in the form of a surrogate key. A link will also contain foreign keys to surrogate keys of the hubs. Additionally, a link contains information about the load date and the source of the data. No information about validity of the data is stored, only past and present relationships between business keys (Williams, 2015).

2.2.2.3 Satellites

Satellites are where the actual descriptive data are stored. Satellites provide context to business objects and tracks changes over time (Williams, 2015). Every version of a business concept will be stored and can be tracked via the load date and load end date.

A satellite is always attached to one link or one hub. The primary key of a satellite is also a foreign key, referring to the surrogate key of the attached link or hub. The validity of a row in the satellite is tracked by a load date and a load end date. When the descriptive data of a business key changes, the previous row in the satellite for that business key will be closed, and a new row containing the new data will be created. The previous data will not be removed, as nothing should be removed from a data warehouse (Linstedt & Olschimke, 2015).

2.2.3 Benefits and limitations

The Data Vault model is capable of handling changes in that it enables structural changes by simply adding more objects and attributes to the model. Having a permanent staging area for source data allows loading data without delays. It also has built in inclusion of metadata, such as data source and loading time. Aspects such as these make the Data Vault model recognized as the optimal choice for DW 2.0 architecture (Jovanovic & Bojicic, 2012).

Though it has substantial benefits, the Data Vault model is not perfect. There are some limitations regarding resilience to change and the tracking of the validity of objects with respect to the real world (Bojičić et al., 2016). Handling the changes is not an issue, as it is mostly done by adding objects to the model. However, this means that the original source used for the model is no longer obtainable, due to irreversible transformation. Secondly, there is no information about the validity of the data. The data are considered valid and active, until a new load provides new metadata (Bojičić et al., 2016).

2.2.4 Data Vault 2.0

In his foreword for Linstedt's book *Building a Scalable Data Warehouse with Data Vault 2.0*, Bill Inmon wrote:

> *Over multiple years, he improved the Data Vault and evolved it into Data Vault 2.0. Today, this System of Business Intelligence includes not only a more sophisticated model, but an agile methodology, a reference architecture for enterprise data warehouse systems, and best practices for implementation. (Linstedt & Olschimke, 2015, p. xv)*

Data Vault 2.0 is based on Data Vault 1.0, but adds methodology and architecture to the already existing data vault modeling. One big change in the data vault model is using hash keys instead of business keys. The hashes are calculated in the staging area. This in turn allows parallel loading, as no key lookup is required between objects. Parallel loading is a substantial performance gain. Hash keys can also be generated for entire rows, which ease the comparison of whether two rows are identical or, whether a row already exists in the target table or not (Linstedt & Olschimke, 2015).

2.3 Cloud computing

There are many various definitions of cloud computing, depending on what aspect is in focus (Abadi, 2009). The National Institute of Standards and Technology provides the following definition:

> *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. (Mell & Grance, 2011, p. 2)*

Usually cloud computing is approached from a business point of view or a technical point of view (Baars & Kemper, 2010). The business view sees cloud computing as *"An Internet based service provision provided on a fee bases" (Baars & Kemper, 2010, p. 1530)*. The technical theme can be described as:

> *A distributed, net-based architecture where resources can be dynamically rearranged. This promises increased cost efficiency (as a result of a higher degree of resource utilization), as well as higher degrees of performance, stability, scalability, and flexibility. (Baars & Kemper, 2010, p. 1530)*

The concept of utilizing cloud computing is becoming more and more appealing. The promise of lower operational costs, higher quality and higher flexibility are one of several motivators for migrating towards cloud-based solutions (Agrawal, Das, & El Abbadi, 2011; Baars & Kemper, 2010; Marston, Li, Bandyopadhyay, Zhang, & Ghalsasi, 2011). Armbrust et al. (2010) predict that cloud computing will grow and become ever more important in the future. Developers need to take scalability into account and make sure that this scaling happens rapidly without delay. The licensing model of paying according to usage for cloud services, suits this requirement very well (Armbrust et al., 2010; Marston et al., 2011).

2.3.1 Privacy and security

Moving data from an on-premises solution to a cloud solution is a very problematic situation. The most radical example of potential issues is the US Patriot Act, that allows the US government to demand access to data stored on a computer or by a third-party hosting provider, in the United States. These data are to be handed over without knowledge or permission of the person/company using the service (Abadi, 2009). According to a study carried out by researchers at a Belgian University, even though US organizations are taking steps to ensure privacy concerns are resolved, there is still a need for considerable improvements (Dhont, Asinari, Poullet, Reidenberg, & Bygrave, 2004). They found that the majority of US organizations have challenges with integrating the Safe Harbor policies into their data processing, regardless of effort and best intentions (Dhont et al., 2004).

It is difficult to ensure that the cloud provider is not using the stored data or even disclosing it, intentionally or by accident. In the cloud, the responsibility for security and privacy is shared among several parties (Armbrust et al., 2010; ComPUtING, 2011).

Another factor impeding the adoption of cloud computing is regulation on a local or national level. Regulation might cover requirements for physical data audit, which becomes problematic when the data are moved to the cloud. Regulation that require SaaS providers to keep data within national or continental boundaries might negate many benefits that cloud computing offer. However, providers are working towards solutions that guarantee a specific data center at a certain geographical location is used to fulfill the requirements mentioned (Marston et al., 2011).

Because cloud computing is still in its infancy, it is reasonable to consider the technological structures immature. The regulation regarding privacy is still not updated to handle data in cloud, and the transfer of data:

> *While the legal issues facing cloud operators and cloud users stem from the fact that personal data is transferred across jurisdictional borders, applicable privacy regulation typically draws a line between data being transferred within an organisation, and data being transferred between organisations. (Svantesson & Clarke, 2010, p.392)*

When a cloud operator transfers data across a border, it remains in the operator's control, and in such cases, privacy regulations regulating transborder data flows might not be applicable (Svantesson & Clarke, 2010).

2.3.2 Benefits and challenges

Cloud computing can be beneficial for many reasons (Krishnan, 2013):

- Infrastructure spending is not up-front, instead it is based on usage. This simplifies budgeting and planning
- Scalability is guaranteed and managed by the provider
- Multiple similar testing and sandbox environments can be configured in parallel, to find the suitable configuration for the production environment.
- Components is reusable and reproducible

Cloud computing can be an advantage when looking at big data. Big data analytics is a challenging task that requires expensive tools, large computational power, and time and effort. Cloud computing can help in solving these problems by providing a scalable resource pool, where only actual usage is charged. Scaling can be done rapidly, both up and down according to the need (Assunção et al., 2015).

Some of the key advantages according to Marston et al. (2011) are:

- Dramatically lowered cost of entry to compute-intensive analytics
- Almost immediate access to hardware, without upfront costs
- Can lower IT barriers to innovation
- On-demand scaling both up and down
- Opens possibilities to new applications and services, that are location-, environment- and context-aware

Even though there are technical solutions and tools, skilled and experienced people are needed to utilize them. Most existing solutions are quite overwhelming to non-experienced users (Assunção et al., 2015).

There is not yet a single perfect solution to manage data in the cloud. Some examples of challenges regarding big data and cloud computing are, how to maintain consistency across different granularities, and how to adjust elasticity for optimizing efficiency regarding costs and resources (Agrawal et al., 2011).

2.3.3 User acceptance

According to a metastudy by Lee, Kozar & Larsen (2003), the Technology Acceptance Model (TAM) by Davis (1985) is still the best model for describing user acceptance determinants.

> *Of all the theories, the Technology Acceptance Model (TAM) is considered the most influential and commonly employed theory for describing an individual's acceptance of information systems. (Y. Lee, Kozar, & Larsen, 2003, p.752)*

By the end of 2017 this doctoral dissertation has been cited over 5 000 times, according to Google Scholar. The TAM proposes that an individual's acceptance of information systems (IS) is determined by the Perceived Usefulness (PU) and Perceived Ease of Use (PEOU). Providing a model for measuring user acceptance with so few variables is revolutionizing within IS research (Lee et al., 2003).

While having an established model and providing a starting point for further extensions and elaborations is crucial, many find that the model has gained too much focus, with little attention directed towards alternative models. Some argue that the oversimplification of TAM is difficult to put into practice, as terms such as useful and easy to use are far too ambiguous (Lee et al., 2003).

Tan and Kim (2015) did a case study on user acceptance of Google Docs, and found that *"Users' confirmation with expectations positively affect their perceived usefulness and satisfaction level.",* and *"Users' perceived usefulness and satisfaction positively affect their intention to continue using such collaboration tools."* (Tan & Kim, 2015, p.439).

Behrend, Wiebe, London, & Johnson (2011) looked at cloud computing adoption and usage in community colleges and found similar results as Tan and Kim. Behrend et al. (2011) found that the ease-of-use and access to alternative tools were directly influencing overall use of cloud computing. Actual usage could not be attributed to the perceived usefulness; however, the perceived usefulness did predict intentions to use cloud computing technologies in the future.

Wu, Lan, & Lee (2013) agree that the use of new technologies such as cloud computing can grant companies a competitive advantage, however if the new technology is met with

low user acceptance it can lead to financial loss and a decline in organizational culture. This echoes the statement by Kimball & Ross (2013) about the importance of the business adopting and using the date warehouse.

Even if many companies are looking to increase their cloud presence, there is still the risk of backlash from internal IT-departments. Some might see the cloud-approach as a threat to their IT-culture, losing control over aspects such as data security and audit policies. Many have a fear of losing their jobs as the internal IT-functions are outsourced. There is also a concern for cloud providers going bankrupt, especially at such a volatile point of technological adoption (Marston et al., 2011).

2.3.4 Data warehousing in the cloud

It is not certain that data warehouses can or will be migrated to cloud infrastructure. Large enterprise data warehouses can reach terabyte and petabyte levels of data, and still need to work under certain time constraints. The latency and bandwidth required is difficult to achieve, even with newer technology and major investments. An alternative approach could be to process and enrich the data in distributed cloud infrastructures and then load these into a physical data warehouse (Baars & Kemper, 2010).

Krishnan (2013) argues that for small and middle-sized businesses, leveraging cloud computing for data warehousing is a valid option. Cloud computing can reduce the costs of keeping the DW running and reduce the need to manage an own internal infrastructure. It also removes the need to invest in updating and upgrading hardware and software.

> *From a business intelligence and data warehouse perspective, the applicability of cloud computing can be extended to reports, analytics, and dashboards. The other areas are still in nascent stages of adoption to cloud computing. (Krishnan, 2013, p. 191)*

2.4 Automation

2.4.1 Definition

In the realm of computer science, automation can be defined as applying automatic control to a process or workflow, or by extension, the use of electronic or mechanical

devices to replace human labor. When considering DWA, the automation could be defined as a full or partial replacement, of a function or task carried out by a human operator (Parasuraman, Sheridan, & Wickens, 2000).

2.4.2 Benefits and challenges

In their research of the evolution of software process, Fuggetta and Di Nitto (2014) state that the most important applications of automation are within configuration management and quality assurance. Configuration management is managed through different version control systems like Git, and collaboration platforms that aim to ease the collaboration and distribution of information regarding the software being developed (Fuggetta & Di Nitto, 2014).

The tools and processes for automating quality assurance have reached a significant maturity, and are being adopted and used in software development at an increasing rate. There are production-ready tools available for automating source code analysis, unit testing and integration testing, acceptance testing, and even the whole testing process including system testing and regression testing (Fuggetta & Di Nitto, 2014).

The purpose of utilizing automation is to increase the capability of human performance and improve safety and efficiency. But automation as a concept can be difficult for people to accept. There are issues such as blindly trusting (misuse) and not trusting and wanting to intervene (disuse) (J. D. Lee & See, 2004). Automation is more than just a technological solution, and this needs to be considered when designing and implementing a product. The feeling of trust is the key component to whether automation will be accepted by the users (J. D. Lee & See, 2004; Sarter, Woods, & Billings, 1997).

> *Automation research emphasizes efficiency, productivity, quality, and reliability, focusing on systems that operate autonomously, often in structured environments over extended periods, and on the explicit structuring of such environments. (Goldberg, 2012, p. 1)*

Even though automated systems have successfully improved efficiency and quality of operations, it is not always without surprises. In many cases, the problem has manifested as a breakdown in the interaction between human operators and the automated system. A typical problematic situation occurs when the operator does not understand what the

automated system is doing, why it is doing it, or what it is going to do next (Sarter et al., 1997).

Sarter et al. (1997) present a few unexpected problems with human-automation interaction. Automation was expected to reduce workloads, but in many cases, it has skewed the distribution of work. The routine work is being managed by automation, but dynamic, time-critical, and high-risk tasks are still managed by human operators (Sarter et al., 1997).

Another example is the increased demands on human operators' knowledge of complex automated systems, and their interaction and programming. Not only must they learn how the system works, but also how to use and control the system, to ensure maximum efficiency and safe operation (Sarter et al., 1997).

> *When new automation is introduced into a system or when there is an increase in the autonomy of automated systems, developers often assume that adding "automation" is a simple substitution of a machine activity for human activity (the substitution myth). Empirical data on the relationship of people and technology suggests that is not the case. (Sarter et al., 1997, p.7)*

3 RESEARCH METHODOLOGY

The thesis is divided into two parts, a theoretical part with a review of the literature in the field of data warehousing and business intelligence, and an empirical part presenting a combination of constructive research and case study research. Because the main point of the research is to implement a real product for the end-customer, an artifact will be constructed according to existing best practices, previous way of working, and scientific literature. After the actual solution is implemented, analysis on how well the artifact corresponded to the actual product will be conducted, as well as how the implementation of the product succeeded.

3.1 Constructive research

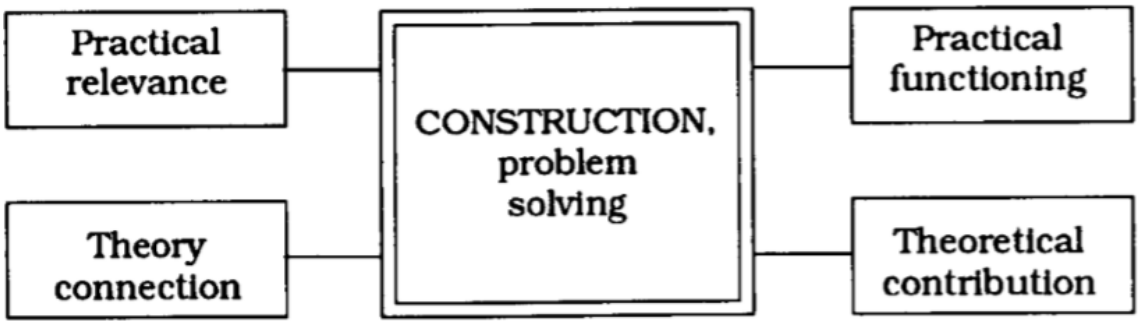


Figure 6 - Elements of Constructive Research (Kasanen, Lukka, & Siitonen, 1993, p.246)

As seen in Figure 6, an essential part of constructive research is to tie the problem and its solution to theoretical knowledge. A general understanding of the problem domain is necessary to assess the applicability of the construct in general.

Kasanen et al. (1993) respond to the critique of problem solving not generating scientific knowledge due to the uniqueness of problems, by arguing that it is *"difficult to imagine a solution to a real-world management accounting problem which would suit well the firm in question but not be suitable to other approximately similar firms."* (Kasanen et al., 1993, p.252-253)

Kasanen et al. (1993) propose that *"A significant share of Master's theses in management accounting follow the plan of problem statement, theory review, a solution of a real-world business problem, and discussion."* (Kasanen et al., 1993, p.244). Even though they target

management accounting research, a parallel can be drawn to information systems research. Kasanen et al. (1993) list some defining features of the constructive method:

- Step by step procedure, with every step defined in the framework.
- It is possible to check every step or phase in the construction.
- The procedure serves a purpose. Building the construction is a goal-oriented activity.

In practice the underlying context for this thesis is problem solving. These features align with the context of this thesis and, as such, the constructive method is a suitable method for approaching the problem.

Iivari (2003) questions the actual core of IS research. On one hand the concept of an IT artefact is central to the discipline. However, Iivari (2003) argues that information systems should form the core, not IT artifacts. Iivari (2003) further suggests that the identity of IS should be based on the view of IS as an applied science, to support IT experts in practice. These arguments support the choice of a constructive method, as the construct will be used in a real-world application.

3.2 Case study research

Benbasat, Goldstein & Mead (1987) define case research through several key characteristics. In case research, typically the phenomenon or problem is examined in its natural setting. According to Benbasat et al., *"research phenomena not supported by a strong theoretical base may be fruitfully pursued through case research."* (Benbasat, Goldstein, & Mead, 1987, p. 372) Case research is innately explorative, because the investigator builds the hypotheses as the complexity of the unit is studied. No dependent or independent variables are defined in advance (Benbasat et al., 1987).

Benbasat et al. (1987) found that the predominant theme in case studies was implementation, and the causes of success or failure. It was rare to have a clearly defined objective, and most studies were characterized as exploratory in nature.

Yin (1981) is a strong proponent of case research. He states that case studies can be done using either qualitative or quantitative evidence. The evidence may consist of *"fieldwork, archival records, verbal reports, observations, or any combination of these."* (Yin, 1981, p.58).

Yin (1981) further states that a case study is not reliant on a particular data collection method. According to him, a case study is a research strategy, comparable to an experiment, history or a simulation. The defining characteristic of a case study is its attempt to examine a contemporary phenomenon in its real-life context, especially in a setting where there are too many variables to account for compared to the number of observations made. This setting renders standard experimental and survey designs irrelevant.

3.3 Application of research methodologies

Because the thesis relies on a real-world business case, constructive research is used to construct an artifact, based on theory, best practices, and previous knowledge and experience. As Kasanen et al. (1993) state, constructive research theses usually follow a plan of defining the problem, reviewing the theory, solving the real-world problem, and discussing the solution, while reflecting on the general applicability of the solution. This statement fits into the context of the thesis very well. The established process of working needs to be modified to suit a cloud-only environment. As much as possible should remain as before, to utilize common practices and keep work as standardized as possible.

The artifact will be implemented as a solution for the end-customer, and as it will be the first of its kind for the company, doing a case study on the project, and its outcome, is well motivated. A case study might reveal fundamental flaws in the construct, but also reveal potential improvements to the strategy of selling standardized and modular solutions to customers.

4 DATA COLLECTION AND ANALYSIS

4.1 SmartEngine

In this chapter, the function and logic of SmartEngine will be presented, along with examples of generated database objects for a simple ER-model with five entities.

The entirety of chapters 4.1.1 & 4.1.2 is based on an interview with the developer of SmartEngine, Kim Johnsson (Johnsson, 2016).

The main reason for the development of SmartEngine was Johnsson's own ambition for reusability and automation. However, the company soon realized that a DWA-tool that removes parts of the manual labor would be a considerable competitive advantage. As presented in chapters 1.4 & 1.4.2, removing parts of the manual labor required to build and maintain a DW has many benefits. Lowering the amount of manual labor needed not only improves quality and reliability through standards enforcement, but also results in a lower cost for the customer, which is an important incentive for potential buyers.

4.1.1 Background and history

SmartEngine is the in-house developed DWA-tool created for implementing a data warehouse based on a data vault model, and maintaining the ETL-procedures and loading of the DW. The tool is not available outside of the company and, as such, is not documented and reviewed.

The developer had a strong background in programming, creating models and taking advantage of models in different circumstances. He has always tried to focus on reusability and finding the minimal effort to complete tasks. In 2000, he became involved in database projects and data warehousing projects. He noticed that there was no reusability between projects at the time. Even if projects had similar models and similar objects, they were still being implemented on a case-by-case basis.

This was a problem, he argued that data warehousing could be compared to a software development project and, therefore, it should be possible to create reusable solutions. The idea was to abstract the concept of loading a certain object into the data warehouse, into loading any kind of object.

There is a recognizable pattern in loading the components of a DW, regardless of content. The Data Vault is well suited for automation, as no data are removed from the DW. All entity types (hubs, links, satellites) always have the same metadata structure, regardless of what business object they represent. The actual descriptive data are stored in the satellites, and do not reference other descriptive data directly, instead relationships are managed through surrogate keys and links.

In 2007, he started to actively consider the problem and come up with a solution. At the time, there were no publicly available automation tools. Even though most of the tools presented in chapter 1.5 existed, they were mostly in-house tools, or not commercially ready. Because of this, Johnsson started developing a tool of his own. One important design decision was to create a tool for implementing models, not just an ETL-tool or a DWA-tool. He wanted SmartEngine to be usable with anything that could be modeled. If the logic was based on rules, and the modeling followed certain standards, anything could be automatically generated.

The first iteration of SmartEngine was capable of creating documentation of a conceptual model. The next iteration was capable of generating the tables in a DW, along with structure and relationships. In 2009, the first prototype was created and presented to TEKES alongside several other internal projects. TEKES granted funding to continue development, and part of this funding was used to further develop SmartEngine. At this point, the city of Turku was involved and a DW project was planned.

In late 2010, SmartEngine managed to automatically load a DW. All procedures and tables were created based on a conceptual model. SmartEngine maintained metadata in a separate database and generated run-time SQL-queries when the load batch was started. The process of calculating business keys, merging and loading of hubs, satellites and links remains unchained to this day.

In 2011, the TEKES funding was spent and the development of SmartEngine was halted. The tool was considered production-ready and was used in the Turku DW-project. The DW was developed and hosted internally. In 2012, the DW was to be migrated to a hosting provider. Afraid of losing trade secrets, the whole logic was re-written. All the run-time queries previously generated using metadata were now created as stored procedures, and no metadata were stored anywhere. This meant the tool could be used to

implement data warehouses, regardless of where it was hosted. At this point the tool only supported SQL Server, due to the company being a Microsoft partner. Technically it would be easy to develop compatibility for other systems, if the need arises.

SmartEngine is designed to function as a platform, and end users of the tool can write their own C# scripts, that utilize the SmartEngine object model. Currently, much of the important logic is embedded in the object model. Johnsson has started developing a new version where all functionality will be in modifiable C# scripts. SmartEngine itself would be nothing more than a platform for scripts. According to Johnsson, this enables the very intriguing possibility of providing the tool as an open-source tool. For a small license fee, others could acquire the platform and write their own scripts, which would be distributed among the community. Johnsson regrets not being more active with the community. He would have wanted to involve TDWI and receive feedback and ideas. One big obstacle has been that the company, being rather small, requires that everyone contributes to revenue. The value of further developing the tool is difficult to measure and, therefore, it has been developed according to a plan based on needs and necessity.

4.1.2 Future plans

In the near future, Johnsson wants to further advance the degree of automation in projects. His mantra is *"If someone can be instructed to do a task, then it is possible to automate said task (Johnsson, 2016)"*. This is a driving force behind all his decisions and plans.

One upcoming feature will be to automate the modeling by importing the structure of source systems and source data and have SmartEngine build a conceptual model based on the data. SmartEngine would be able to find primary keys, business keys and foreign keys. When not automatically detected, the developer could add metadata using a visual UI, removing the need for a separate modeling tool.

Another feature is the ability to have SmartEngine generate fact-tables and dimension-tables. The developer needs to add metadata on rules for the data, on what granularity the data will be aggregated, should date ranges be expanded to individual months, days etc.

Johnsson already has a long-term plan in mind. Perhaps the most important is to help with creating and managing master data. It is common to have some master data that are manually updated using something like Excel. SmartEngine could generate a form or

application to help with this, and support the loading of said data, without the need for manual labor in the staging area.

Johnsson also wants to incorporate Big Data. Big Data eventually has a structure, which means it can be modeled at some point. All it takes is to create the rules according to which the data will be handled. Johnsson points out that SmartEngine is not only a DWA-tool, it is also a tool for implementing any model. Another important feature is to support enriching of data. There needs to be a simple way to enrich source data with open data and big data.

4.1.3 Function and logic

Because SmartEngine is built to work on an exported ER-model of a certain format, it is reliant on Visual Paradigm. The export contains all entities, relationships, attributes and metadata, in XML-format (Extensible Markup Language). Any modeling tool could be used, but then the logic of the application would need to be tweaked to accommodate the alternate export of the model.

SmartEngine uses this export to create an internal object model to be used when generating the necessary SQL scripts and database objects. The rules and logic for generating the database objects will be presented in chapter 4.2.

Having a separate object model enables developers to write additional scripts for SmartEngine using C#. The object model of SmartEngine will always be standardized, regardless of what the ER-model looks like. Any changes in the source code are automatically reflected in the object model documentation. The idea is that scripts for SmartEngine are collaborative and iterated upon. Anything a developer makes could potentially be usable by the rest of the team. It allows the SmartEngine-developer to focus on updating and further improving the tool, and the DW-developers can create a script for what they need or want.

4.1.4 Loading the Data Vault

When loading a data vault, the hubs are loaded first. As seen in Figure 7, loading a hub is quite simple. All business keys must be unique in the hub and have a unique surrogate key. Hubs are not time-dependent, rows in a hub are never ended. Once a business key

has been loaded, it will remain in the hub even if it is never loaded again. All new business keys will be inserted, and the metadata for each business key present in a load batch will be updated (Linstedt & Olschimke, 2015). The structure of hubs is presented in chapter 4.2.3.

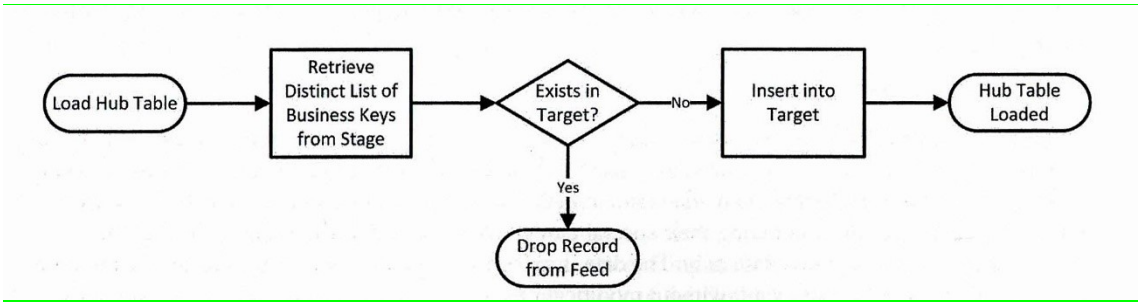


Figure 7 - Template for loading a hub table (Linstedt & Olschimke, 2015, p. 434)

Once the hubs have been loaded, all links and satellites can be loaded in parallel. Figure 8 illustrates the steps in loading a link table. Because a link is modeled in a certain direction from one hub to another hub, there can be several rows in the link table for each source hub. Because of this, we need to maintain metadata about the validity of rows. Each row in the hub can only have one active referring row in the link. Because of this, when a new row is inserted for a business key, any rows not present in the load batch for that business key will be ended (Linstedt & Olschimke, 2015). The structure of links is examined more closely in chapter 4.2.5.

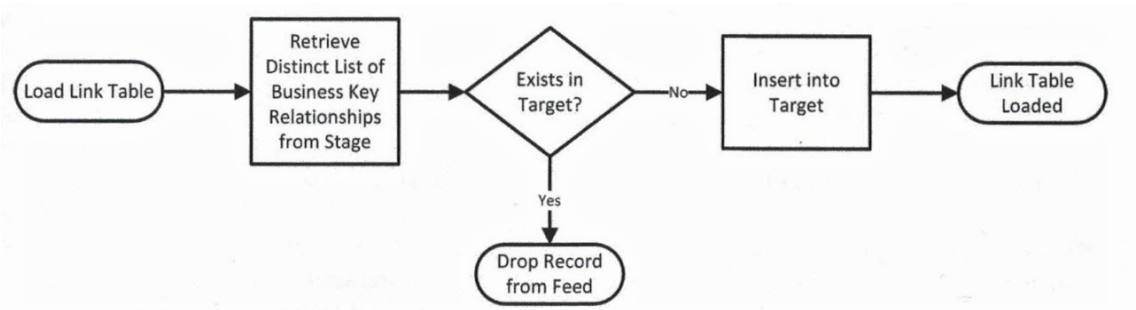


Figure 8 - Template for loading a link table (Linstedt & Olschimke, 2015, p. 449)

Loading the satellites involves slightly more work, as seen in Figure 9. When inserting new rows into the satellite, not only must the surrogate key of the hub be found, but every column of descriptive data must be compared with data already in the satellite table. If there is no row with exactly the data found in the staging area, it will be inserted. After insertion, all rows missing from the load batch for a loaded business key must be ended,

as with the links. For any given time, there can only be one active row in the satellite table, per business key in the hub.

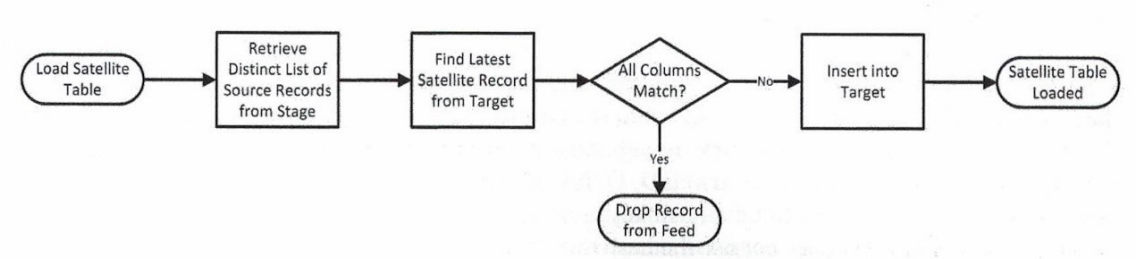


Figure 9 - Template for loading a satellite table (Linstedt & Olschimke, 2015, p. 469)

4.2 Database objects generated by SmartEngine

SmartEngine is designed to create a data warehouse according to Data Vault 1.0, described in chapter 2.2. In this chapter, we will briefly examine the generated database objects and what rules they follow. To illustrate, a simple ER-model with five entities will be used to present the generated DW-objects.

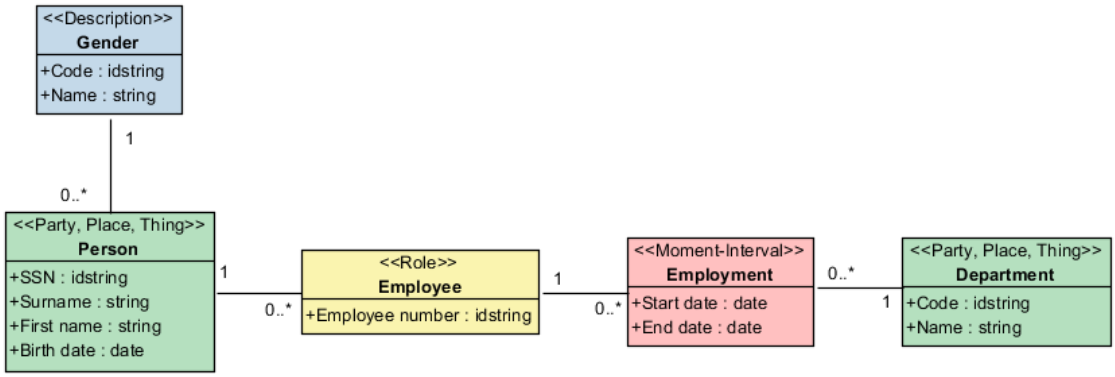


Figure 10 - Example of five entities in an ER-model (Screenshot from Visual Paradigm, Eklund, 05.01.2018)

Figure 10 contains five entities, with direct relationships. A person always has one gender, and for any gender there can be zero or more persons. A person can be an employee, and can even be several different employees in the system, because his employee number might change if he has been absent from the company for some years. An employee can have zero or many employments, and an employment always has one employee. The employment is to a certain department, and a department can have several employments.

The entity Employment is necessary because an employee can belong to several departments over time, and departments can have several employees. This is a many-to-many relationship that cannot be modeled. This is solved by placing a time-related role entity in between.

The model is not complete and is missing many entities and several attributes, but it will be used to present the objects generated by SmartEngine and their structure in the database.

For a person, we have the attributes SSN (Social Security Number), first name, surname and birth date. For gender, we have a code and a name. For employee we have an employee number. For employment we have a start date and an end date. A more thorough examination of the ER-modeling standard the company uses, will be presented in chapter 4.3.2.

Being a Finnish company, most customers are Finnish and, therefore, the ER-models are primarily in Finnish. Internal column names will mostly be in Finnish, however some terms related to the Data Vault model theory are in English.

4.2.1 **OUT-tables**

For each entity in the model, an OUT-table is created in the staging area. The OUT-table will be the source for the loading procedures that SmartEngine generates. This implies that the source data needs to be inserted into the OUT-table, into the correct columns. Alternatively, the table can be replaced by a view with the same name, with the same structure as the generated table had.

Standard practice is to have all the columns in an OUT-table have the data type nvarchar. SmartEngine converts to the correct data type when loading the storage layer.

For the entity Person, an OUT-table with the following columns will be generated:

Fieldname	Description	Origin
ID	**Primary key**. This is the column used to identify each row and, therefore, must be unique. It can be either a value from the source data if it contains a uniquely identifying value, else it is a compound of the other columns from the source data. In this example, the SSN would be a good candidate	Source data
Kommentti	A string column for miscellaneous comments related to the data warehouse, for the developers	Manual
Tietolahde	A string column describing the origin of data	Manual
SSN	An attribute of the entity	Source data
Surname	An attribute of the entity	Source data
FirstName	An attribute of the entity	Source data
BirthDate	An attribute of the entity	Source data
GenderId	A **foreign key**, pointing to the primary key of the entity Gender. It originates from source data, but must point to an existing primary key, otherwise the loading procedures will not be able to load the entity correctly	Source data

Table 1 - The OUT-table for the entity Person

For the entity Gender, an OUT-table with the following columns will be generated:

Fieldname	Description	Origin
Id	**Primary key**, in this case the code would be a good candidate	Source data
Kommentti	A string column for miscellaneous comments	Manual
Tietolahde	A string column describing the origin of the source data	Manual
Code	A descriptive attribute of the entity	Source data
Name	A descriptive attribute of the entity	Source data

Table 2 - The OUT-table for the entity Gender

For the entity Employee, the OUT-table will have the following structure:

Fieldname	Description	Origin
Id	**Primary key**, in this case the employee number would be a good candidate	Source data
Kommentti	A string column for miscellaneous comments	Manual
Tietolahde	A string column describing the origin of the source data	Manual
EmployeeNumber	A descriptive attribute of the entity	Source data
PersonId	A **foreign key**, pointing to the primary key of the entity Person	Source data

Table 3 - The OUT-table for the entity Employee

The entity Employment will have the following structure:

Fieldname	Description	Origin
Id	**Primary key**, in this case there is no clear choice as the entity is created to solve a many-to-many relationship in the model. Therefore, a combination of columns to uniquely identify the row will be used	Source data
Kommentti	A string column for miscellaneous comments	Manual
Tietolahde	A string column describing the origin of the source data	Manual
StartDate	A descriptive attribute of the entity	Source data
EndDate	A descriptive attribute of the entity	Source data
EmployeeId	A **foreign key**, pointing to the primary key of the entity Employee	Source data
DepartmentId	A **foreign key**, pointing to the primary key of the entity Department	Source data

Table 4 - The OUT-table for the entity Employment

The entity Department will be generated with the following structure:

Fieldname	Description	Origin
Id	**Primary key**, in this case the code would be a good candidate	Source data
Kommentti	A string column for miscellaneous comments	Manual
Tietolahde	A string column describing the origin of the source data	Manual
Code	A descriptive attribute of the entity	Source data
Name	A descriptive attribute of the entity	Source data

Table 5 - The OUT-table for the entity Department

4.2.2 ETL-tables

SmartEngine keeps track of loads and timepoints through a table called ETLLoad. The table contains three columns:

Fieldname	Description	Origin
Id	The LoadId, whenever a point-in-time is referenced in the storage layer, it is to a certain LoadId found in this table. Utilizes the SQL Server's column property Identity[1].	SmartEngine
StartTime	The timestamp for when a LoadId was generated	SmartEngine
EndTime	The timestamp for when a LoadId was finished. Will be null until the load batch has ended	SmartEngine

Table 6 - The table ETLLoad, for keeping track of load batches

Whenever a new load is started, the next available LoadId will be declared. Once all loading procedures related to that batch has finished, the LoadId will receive its ending timestamp.

SmartEngine relies on global temporary tables when running the load procedures. A temporary table will be created for each entity that is to be loaded. This is where the Business Key for an entity will be calculated, and where the conversion of attributes to the correct data type will occur. The entity's OUT-table is used as source.

[1] https://docs.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql-identity-property

For the entity Person, we have the following columns:

Field Name	Description	Origin
Id	The **primary key**, from OUT_Person	Source data
Kommentti	The comment column	Manual
Tietolahde	The origin of the source data	Manual
SSN	The attribute SSN converted to the correct data type	SmartEngine
SSN_ORIG	The original attribute SSN without transformation	Source data
Surname	The attribute surname converted to the correct data type	SmartEngine
Surname_ORIG	The original attribute surname without transformation	Source data
FirstName	The attribute first name converted to the correct data type	SmartEngine
FirstName_ORIG	The original attribute first name without transformation	Source data
BirthDate	The attribute birth date converted to the correct data type	SmartEngine
BirthDate_ORIG	The original attribute birth date without transformation	Source data
GenderId	A **foreign key**, pointing to the primary key of the entity Gender	Source data
BusinessKey	The calculated business key for the entity Person	Source data

Table 7 - The ETL-table for the entity Person

The ETL-table for the entity Gender will consist of the following columns:

Field Name	Description	Origin
Id	The **primary key**, from OUT_Gender	Source data
Kommentti	The comment column	Manual
Tietolahde	The origin of the source data	Manual
Code	The attribute code converted to the correct data type	SmartEngine
Code_ORIG	The original attribute code without transformation	Source data
Name	The attribute name converted to the correct data type	SmartEngine
Name_ORIG	The attribute name without transformation	Source data
BusinessKey	The calculated business key for the entity Gender	Source data

Table 8 - The ETL-table for the entity Gender

The ETL-table for the entity Employee will consist of the following columns:

Fieldname	Description	Origin
Id	The **primary key**, from OUT_ Employee	Source data
Kommentti	The comment column	Manual
Tietolahde	The origin of the source data	Manual
EmployeeNumber	The attribute employee number converted to the correct data type	SmartEngine
EmployeeNumber_ORIG	The original attribute employee number without transformation	Source data
PersonId	A **foreign key**, pointing to the primary key of the entity Person	Source data
BusinessKey	The calculated business key for the entity Employee	Source data

Table 9 - The ETL-table for the entity Employee

The ETL-table for the entity Employment will consist of the following columns:

Fieldname	Description	Origin
Id	The **primary key**, from OUT_ Employment	Source data
Kommentti	The comment column	Manual
Tietolahde	The origin of the source data	Source data
StartDate	The attribute start date converted to the correct data type	SmartEngine
StartDate_ORIG	The original attribute start date without transformation	Source data
EndDate	The attribute end date converted to the correct data type	SmartEngine
EndDate_ORIG	The original attribute end date number without transformation	Source data
EmployeeId	A **foreign key**, pointing to the primary key of the entity Employee	Source data
DepartmentId	A **foreign key**, pointing to the primary key of the entity Department	Source data
BusinessKey	The calculated business key for the entity Employment	Source data

Table 10 - The ETL-table for the entity Employment

The ETL-table for the entity Department will consist of the following columns:

Fieldname	Description	Origin
Id	The primary key, from OUT_ Department	Source data
Kommentti	The comment column	Manual
Tietolahde	The origin of the source data	Manual
Code	The attribute code converted to the correct data type	SmartEngine
Code_ORIG	The original attribute code without transformation	Source data
Name	The attribute name converted to the correct data type	SmartEngine
Name_ORIG	The attribute name without transformation	Source data
BusinessKey	The calculated business key for the entity Department	Source data

Table 11 - The ETL-table for the entity Department

If the business key of an entity A is also a part of the business key of entity B, the business key of entity A must be calculated first. SmartEngine interprets this from the ER-model

and forces the business key calculation to happen in the correct order. The method of doing this will be presented later, in chapter 4.2.7.

4.2.3 Hubs

SmartEngine generates hubs according to Data Vault theory, as presented in chapter 2.2.2.1. Every entity will receive a hub. The hub contains a running identification number functioning as a surrogate key, the business key, and metadata about when the business key was seen, referring to a certain LoadId as described earlier. All generated hubs will have the following columns:

Fieldname	Description	Origin
Id	The surrogate key, the running identification number for hubs	SmartEngine
Avain	The business key	Source data
Kasite	The name of the entity	SmartEngine
Tietoluotu	The first LoadId where the business key appeared.	SmartEngine
TietoHavaittu	The latest LoadId where the business key appeared	SmartEngine
Kommentti	The comment column	Manual

Table 12 - The structure of any hub-table in the DW

4.2.4 Satellites

Satellites are generated according to Data Vault theory, as presented in chapter 2.2.2.3. The satellites contain all versions of the attributes for each business key. If the data changes, the satellite will receive a new row for that business key. The satellite also has columns for the validity of a row, referring to LoadIds. Only one row should ever be active. This enables the DW to store all versions of the source data, and still only present the valid version. This means that for any foreign key to a single business key, there can be several rows in the satellite, which means several surrogate keys.

For the entity Person, SmartEngine will create a table with the following columns:

Fieldname	Description	Origin
Id	The surrogate key, a running identification number for each satellite	SmartEngine
HubiId	A foreign key, referring to the surrogate key of the hub for the entity Person	SmartEngine
Kommentti	The comment column from the OUT-table	Manual
TiedonVoimassaoloAlku	The first LoadId where these data were valid	SmartEngine
TiedonVoimassaoloLoppu	The last LoadId where these data were valid. This value can be null if the row is still valid	SmartEngine
SSN	The attribute SSN	Source data
Surname	The attribute surname	Source data
FirstName	The attribute first name	Source data
BirthDate	The attribute birth date	Source data

Table 13 - The satellite-table for the entity Person

The entity Gender will receive a satellite with the following columns:

Fieldname	Description	Origin
Id	The surrogate key, a running identification number for each satellite	SmartEngine
HubiId	A foreign key, referring to the surrogate key of the hub for the entity Gender	SmartEngine
Kommentti	The comment column from the OUT-table	Manual
TiedonVoimassaoloAlku	The first LoadId where these data were valid	SmartEngine
TiedonVoimassaoloLoppu	The last LoadId where these data were valid. This value can be null if the row is still valid	SmartEngine
Code	The attribute code	Source data
Name	The attribute name	Source data

Table 14 - The satellite-table for the entity Gender

The entity Employee will receive a satellite with the following columns:

Fieldname	Description	Origin
Id	The surrogate key, a running identification number for each satellite	SmartEngine
HubiId	A foreign key, referring to the surrogate key of the hub for the entity Employee	SmartEngine
Kommentti	The comment column from the OUT-table	Manual
TiedonVoimassaoloAlku	The first LoadId where these data were valid	SmartEngine
TiedonVoimassaoloLoppu	The last LoadId where these data were valid. This value can be null if the row is still valid	SmartEngine
EmployeeNumber	The attribute employee number	Source data

Table 15 - The satellite-table for the entity Employee

The entity Employment will receive a satellite with the following columns:

Fieldname	Description	Origin
Id	The surrogate key, a running identification number for each satellite	SmartEngine
HubiId	A foreign key, referring to the surrogate key of the hub for the entity Employment	SmartEngine
Kommentti	The comment column from the OUT-table	Manual
TiedonVoimassaoloAlku	The first LoadId where these data were valid	SmartEngine
TiedonVoimassaoloLoppu	The last LoadId where these data were valid. This value can be null if the row is still valid	SmartEngine
StartDate	The attribute start date	Source data
EndDate	The attribute end date	Source data

Table 16 - The satellite-table for the entity Employment

The entity Department will receive a satellite with the following columns:

Fieldname	Description	Origin
Id	The surrogate key, a running identification number for each satellite	SmartEngine
HubiId	A foreign key, referring to the surrogate key of the hub for the entity Department	SmartEngine
Kommentti	The comment column from the OUT-table	Manual
TiedonVoimassaoloAlku	The first LoadId where these data were valid	SmartEngine
TiedonVoimassaoloLoppu	The last LoadId where this data was valid. This value can be null if the row is still valid	SmartEngine
Code	The attribute code	Source data
Name	The attribute name	Source data

Table 17 - The satellite-table for the entity Department

4.2.5 Links

Links will be generated when there is a direct relationship between two hubs, according to the theory presented in chapter 2.2.2.2. In the example case, there will be a link generated, from person to gender. Every person is modeled to have one gender. This will result in the creation of a link for the entity Person.

The link-table consists of the following columns:

Fieldname	Description	Origin
Id	The surrogate key, a running identification number for each link	SmartEngine
HubiId	A foreign key, referring to the surrogate key of the hub for the entity Person	SmartEngine
Kommentti	The comment column from the OUT-table	Manual
TiedonVoimassaoloAlku	The first LoadId where these data were valid	SmartEngine
TiedonVoimassaoloLoppu	The last LoadId where these data were valid. This value can be null if the row is still valid	SmartEngine
GenderId	A foreign key, pointing to the surrogate key of the hub of Gender	Source data

Table 18 - The link-table for the entity Person

As mapped in the OUT-table for person, every business key in the entity Person will point to a business key in the entity Gender. The link for Person will maintain this information and all previous versions of it, while presenting only one link as valid at a time per business key.

Two other links will be generated, a link for Employee and a link for Employment. An employee is always a person, so the link will be from Employee to Person. The link-table consists of the following columns:

Fieldname	Description	Origin
Id	The surrogate key, a running identification number for each link	SmartEngine
HubiId	A foreign key, referring to the surrogate key of the hub for the entity Employee	SmartEngine
Kommentti	The comment column from the OUT-table	Manual
TiedonVoimassaoloAlku	The first LoadId where these data were valid	SmartEngine
TiedonVoimassaoloLoppu	The last LoadId where these data were valid. This value can be null if the row is still valid	SmartEngine
PersonId	A foreign key, pointing to the surrogate key of the hub of Person	Source data

Table 19 - The link-table for the entity Employee

The link for Employment will be slightly different. Because it models a many-to-many relationship, the link-table will have two foreign keys. As the satellite contains the start and end date of the employment, the combination of satellite and link will return the exact pairing that was valid at a certain point of time.

The structure of the link table for the entity Employment will be the following:

Fieldname	Description	Origin
Id	The surrogate key, a running identification number for each link	SmartEngine
HubiId	A foreign key, referring to the surrogate key of the hub for the entity Employment	SmartEngine
Kommentti	The comment column from the OUT-table	Manual
TiedonVoimassaoloAlku	The first LoadId where these data were valid	SmartEngine
TiedonVoimassaoloLoppu	The last LoadId where these data were valid. This value can be null if the row is still valid	SmartEngine
EmployeeId	A foreign key, pointing to the surrogate key of the hub of Employee	Source data
DepartmentId	A foreign key, pointing to the surrogate key of the hub of Department	Source data

Table 20 - The link-table for the entity Employment

4.2.6 K-views

SmartEngine automatically generates a view for each entity in the ER-model to make it easier to use the data vault. These views are called K-views. The K-view is not part of data vault theory, it is an addition to SmartEngine. The K stands for “käsitemalli”, which is Finnish and translates to “conceptual model”. The K-views are meant to be used for the creation of data marts, fact-tables, and dimensions. The SQL-code for K_Person is as follows:

```
CREATE view [dw].[K_Person] as
select
            fact.HubiId as Id,
            fact.SSN,
            fact.Surname,
            fact.FirstName,
            fact.BirthDate,
            link.GenderId
from
            dwData.S_Person as fact
            inner join dwData.L_Person as link
                        on fact.HubiId = link.HubiId
                        and link.TiedonVoimassaoloLoppu is null
                        and fact.TiedonVoimassaoloLoppu is null
```

Most of the data presented in the view come from the satellite of person, with the addition of the surrogate key of gender's hub. The developer is not required to utilize the storage layer of the DW, as the K-views do the combining automatically. The two last conditions of the join-statement ensure that only valid data are queried. As previously mentioned, if the "end of validity"-column is null, it means that the data are still valid.

It is possible to add a flag for denormalization when modeling the entities. For each flagged entity, SmartEngine will generate a procedure that queries the attribute data of each entity directly linked to the flagged entity in the ER-model. It will become a table with the postfix _DN attached. In the case of the example model, a table called K_Person_DN will be generated with the following structure:

Fieldname	Description	Origin
Id	The surrogate key of person's hub	Person's hub
SSN	The attribute SSN	Person's satellite
Surname	The attribute surname	Person's satellite
FirstName	The attribute first name	Person's satellite
GenderId	The surrogate key of the linked hub of gender	Gender's hub
GenderCode	The attribute code	Gender's satellite
GenderName	The attribute name	Gender's satellite

Table 21 - The denormalized table for the entity Person

4.2.7 Procedures

SmartEngine generates a large number of stored procedures to handle the transformation and loading. Every entity will have its own business key calculation procedure, and every hub, satellite, and link will have its own loading procedure. These are generated into an ETL-schema, usually called etl. The name is configured as metadata for the ER-model.

SmartEngine also generates a few high-level ETL-procedures. Two procedures are generated for starting and finishing a load batch. These manage the declaration of a new LoadId, and its timestamps for start and finish of the load batch. A third procedure is generated, that controls the execution order of the loading procedures.

The procedure for business keys calculation includes some validation. Looking at the entity person, the first step of its procedure checks whether person's business key has been calculated, due to another entity depending on person's business key. If the ETL-table has rows, then the calculation has already happened and no further action will be taken. If not, then the next step is to take the rows from OUT_Person, do the necessary casting and conversion to correct data types and then insert them into the ETL-table. Once that is done, the procedure performs validation by checking if the table contains duplicate business keys, or duplicate primary keys. Duplicates are removed, so the loading and merging can complete, but this entails that not all data will be loaded.

The procedure for loading hubs begins by calling on the business key calculation. If the hub is empty, then all rows from the ETL-table are simply inserted. If rows exist, the procedure uses the SQL-function merge()[2] to add the missing rows, and then updates the metadata of when each business key was last seen.

The procedure that executes the loading procedures in the correct order is called RunFromSchema. The logic is quite simple, it looks for all procedures in a certain schema that match the prefix "Load_". These are ordered alphabetically in a cursor. The loading procedures all follow the same naming convention. For instance, for the entity Person the following procedures will be generated:

- Etl.Load_H_Person
- Etl.Load_L_Person
- Etl.Load_S_Person

Because all procedures are gathered in the SQL-cursor, it will result in all hubs being loaded first, then all the links, and lastly all the satellites. The order in which all hubs are loaded does not matter, as they do not refer to each other. Links connect hubs, and because the hubs are forcibly loaded first and any potentially new surrogate keys generated, the internal order of which links are loaded in, does not matter. Once all the links have been loaded and their potentially new surrogate keys created, the satellites can be loaded.

[2] https://docs.microsoft.com/en-us/sql/t-sql/statements/merge-transact-sql

Because hubs are not directly joined to each other, they can be loaded in parallel. Links describe relationships between hubs, and can also be loaded in parallel once the hubs have been loaded. Satellites cannot be attached to each other, only to hubs and links and, therefore, they can be loaded in parallel as well. There can be long dependency chains, if one or several business keys are part of another entity's business key. This entails that all involved business keys must be calculated recursively, which often results in a slowdown. However, once a business key is calculated, any following calls to calculate said business key will be skipped. No business key is ever calculated twice, and unnecessary business keys are not calculated, in case the load batch is a partial load.

In the example case, the order in which loading procedures are executed is:

1. Load_H_Department
2. Load_H_Employee
3. Load_H_Employment
4. Load_H_Gender
5. Load_H_Person
6. Load_L_Employee
7. Load_L_Employment
8. Load_L_Person
9. Load_S_Department
10. Load_S_Employee
11. Load_S_Employment
12. Load_S_Gender
13. Load_S_Person

SmartEngine contains further functionality and capability, but they are not used as often, and are not relevant in the context of this thesis.

4.3 Previous process of implementing a data warehouse

The steps of the process will be separated into parts according to the definition in Chapter 2.1.1. In the following chapters, the established way of working at the company will be presented, when implementing a traditional on-premises data warehouse. Both the process and tools used will be presented.

As mentioned in the introduction, the company is a certified Microsoft partner and, because of this, relies on Microsoft tools and solutions. The implementation of the artifact

will very likely be based on Microsoft tools and platforms, due to it being a real project with a real customer.

4.3.1 Tools and services used in the process

Visual Paradigm

Visual Paradigm (VP) is an enterprise management and software development suite. It contains tools for managing all steps and tasks in the design and development of a project or product. It also contains tools for project management and team collaboration. The company has only used the modeling capabilities, and now VP is only used for ER-modeling. In the past, VP has been used to model use cases, test cases, processes, and different workflows and data flows.

The tool was chosen out of several candidates in 2006, and there were several reasons for choosing VP. It had a very good benefit-cost ratio, was easily expandable to suit the requirements of SmartEngine, and had a simple way of modeling and modifying existing models. It was one of the few tools that had the capability to export the entire model, with all metadata, to an XML-file. It has features called “stereotypes” and “tagged values”, that are used to store metadata for SmartEngine.

VP has an embedded version control, which simultaneously enables team collaboration. ER-models can be branched and merged, much like with standard version control systems. Each change to a model is pushed as a commit, and a model can be reverted to any previous commit.

SmartEngine

SmartEngine is used throughout the DW development phase. The functionality and design of SmartEngine is discussed in chapters 4.1 and 4.2.

SQL Server

SQL Server is a relational and operational database management system, developed by Microsoft. The first version of SQL Server, SQL Server 1.0, rolled out in 1989 for OS/2. There have been many different versions and editions along the years, with SQL Server 2017 being the newest version at the time of writing.

Queries are written in a language called T-SQL (Transact-SQL). It contains some extensions to the SQL standard, but is very similar to the SQL of MySQL and others.

SQL Server includes separate modular services that add functionality to the server. None are required for database functions, rather they add value on top of the database management system. The ones central to data warehousing are Integrations Services, Reporting Services and Analysis Services. These will be presented later in this chapter.

SQL Server contains a SQL Server Agent, which is used for scheduling and automating batches of tasks. The agent can execute a variety of tasks, such as running SQL-queries, running stored SQL procedures, executing SSIS packages and executing Analysis Services commands and queries.

SQL Server Management Studio

Almost all database development is done using SQL Server Management Studio (SSMS). SSMS is a SQL-development environment created by Microsoft. Microsoft describes SSMS as:

> *SQL Server Management Studio (SSMS) is an integrated environment for managing any SQL infrastructure. Use SSMS to access, configure, manage, administer, and develop all components of SQL Server, Azure SQL Database, and SQL Data Warehouse. SSMS provides a single comprehensive utility that combines a broad group of graphical tools with a number of rich script editors to provide access to SQL Server for developers and database administrators of all skill levels. (Microsoft, 2017b)*

With SSMS it is possible to develop and manage the database objects, and configure and maintain the SQL Server infrastructure. Most of the DW-development is done using this tool.

SQL Server Data Tools

SQL Server Data Tools (SSDT) is tool that is based on Visual Studio. It can be installed as an extension to an existing Visual Studio, or it can function as a stand-alone application, which installs a minimal shell of Visual Studio.

SSDT is used to develop and deploy Integration Services packages, Analysis Services data models, and Reporting Services reports. It also includes functionality to develop and deploy SQL Server Relational databases and Azure SQL databases, but that is usually done through SSMS.

SQL Server Integration Services

SQL Server Integration Services (SSIS) is a platform by Microsoft, for building data integration and data transformation solutions. SSIS can handle a wide variety of sources and destinations, such as XML files, flat files, Excel files, relational data sources, and cloud storage solutions. With free feature packs, additional sources and destinations can be supported, for instance, Azure Storage, Hadoop, and HDFS.

SSIS includes built-in tasks and transformations, and has a graphical user interface that allows integration solutions to be developed without writing code. The various built-in tasks in SSIS can be extended by writing C#-scripts and tasks. SSIS supports file system tasks which makes archiving loaded files easy. SSIS variables enable reusability of tasks and flows, through wildcard support for filenames and folders.

Integration Services is a part of the SQL Server package and has a service that can manage and run packages and solutions deployed to the SQL Server. SSIS projects are authored in SQL Server Data Tools for Visual Studio.

SQL Server Analysis Services

> *Analysis Services is an analytical data engine used in decision support and business analytics, providing the analytical data for business reports and client applications such as Power BI, Excel, Reporting Services reports, and other data visualization tools.*
>
> *A typical workflow includes authoring a multidimensional or tabular data model, deploying the model as a database to an on-premises SQL Server Analysis Services or Azure Analysis Services server instance, setting up recurring data processing, and assigning permissions to allow data access by end-users. When it's ready to go, your semantic data model can be accessed by any client application supporting Analysis Services as a data source. (Microsoft, 2017c)*

SQL Server Analysis Services provides OLAP and data mining capabilities. It is mostly used to serve as the semantic layer for the reports and end-users. Having this layer allows uninterrupted use of reports when the data warehouse is being loaded, or when the data marts are being updated. The data models are authored in SQL Server Data Tools (SSDT) for Visual Studio. When using tabular data models, Data Analysis Expressions (DAX) are used to build formulas and expressions to fulfill reporting needs. In the case of multidimensional data models, the language used is Multidimensional Expressions (MDX). Microsoft recommends the use of tabular models, as they are better suited for the latest self-service BI tools and cloud services (Microsoft, 2017a). Russo (2014) also recommends tabular models, from previous experience with helping companies choose between tabular and multidimensional:

> *The reasons why Tabular was preferred to Multidimensional are performance, maintenance, cost of ownership, and flexibility in data model design. This flexibility is very important considering the requirements to automate the provisioning of the data model (Russo, 2014).*

SQL Server Reporting Services

SQL Server Reporting Services (SSRS) provides a report generation environment, designed to deliver a variety of different reports to end-users. Usually the reports made with SSRS are static, pre-defined, enterprise-level reports. Parameter options can be provided for customization or drill-down of the reports. The reports can be scheduled to refresh and be delivered, on a specified schedule.

The reports can use relational or multidimensional data sources. SSRS is well integrated with SQL Server tools and modules. The reports can be developed in Visual Studio, or in a separate application, Report Builder. Reports can be viewed with any internet browser, or they can be embedded in a SharePoint site.

Excel

Microsoft Excel is a spreadsheet software with features for calculation and formulas, graphing, pivot tables, and a programming language called Visual Basic for Applications (VBA). It has long been used in statistical domains, financial domains, and several other domains. Excel has good connectors that allow data retrieval from databases or SSAS cubes, and can present the data in pivot tables and graphs. Excel can create an internal data model, through an add-in called Power Pivot. The data model can consist of related tables, as with SSAS data cubes. Power Pivot supports measures, hierarchies, KPIs, calculated columns, and much more.

Because Excel is widely used in organizations and many end users feel comfortable using Excel, it has a solid foundation as a BI reporting tool, and provides ad hoc analysis capability for the advanced end-user.

Power BI

Power BI is a collection of tools meant to deliver insights into an organization's data. Power BI provides means to prepare the data, analyze and visualize them, and publish to the entire organization for consumption. It supports consumption of prebuilt reports and dashboards, but also allows end-users to do ad hoc analysis and make their own copies of the published reports.

Power BI is a cloud-based business intelligence and analytics platform. Power BI supports a wide variety of data sources, even by other non-Microsoft providers and technologies, and more sources are added monthly. With Power BI, an internal data model with measures and expressions can be created using DAX, similar to tabular data models made with SSAS. The internal data model enables combining of data sources for reports, even without a data warehouse and reporting layer. Power BI includes support for custom visuals, advanced mapping, row-based security, and scheduled delivery of reports.

Reports are designed and created in Power BI Desktop, a free desktop application. If an internal semantic data model is needed, that too is implemented in Power BI Desktop. The reports are normally published to the Power BI portal, found at powerbi.com. A gateway to the data source can be installed and configured, to keep the data on-premises.

Alternatively, a Premium tenant can be acquired which enables installation of a Power BI Report Server on a local server inside the organization.

There are two different ways to access data with Power BI Desktop, import and live connection. Importing means pulling the data to Power BI's internal data model and publishing it along with the report. The data can be scheduled to refresh at certain intervals. When importing, it is possible to make changes to the data and to create additional measures and calculations. Reports created this way are limited to 1 GB of data. It is worth to note that when creating reports this way, the data will permanently be stored in the cloud.

The alternative is to use a live connection to the data source. This means that anyone consuming the report will cause queries to be run at the data source. The benefit is that the data are temporarily stored in the cloud and removed once the report is closed. In many cases, a data gateway is necessary to access the data source through the report, if direct access to the data source is not available.

4.3.2 ER-modeling

The company uses a tool called Visual Paradigm for modeling purposes. The goal of conceptual modeling is to describe entities of interest in a specific domain, and how they are related to each other. The model must adhere to the business processes and be an accurate description of the whole being modeled.

A correct and complete conceptual model is crucial to the success of a DW-project. Several types of expertise are required to ensure that the model is robust, caters to the business needs, and is technically reliable and long-lasting.

The most important aspect is the business knowledge and domain knowledge. What are the Key Performance Indicators based on? What are the specifics of a certain domain, what is tracked and monitored? How can the flow or process be modeled? Related to these, is the technical knowledge of the customer. What are the source systems, how are they designed? How does data entry fit into the process? What is the end-result they are using now? Customers have a varying degree of understanding of the technical side of their work, but usually the business knowledge is solid.

It is also important to teach the customer about modeling, why it is done, and what needs to be considered. There are technical limitations to what DWA-tools like SmartEngine can do. The prior experience helps spot potential pitfalls and bottlenecks in the model, and eventually in the design of the data warehouse. Any modifications and clarifications should be done as early as possible.

At this stage, the roles of the business analyst and product owner are critical. They should be the link between customer and developer. They need to understand the customer's problem domain, but also understand the technical limitations and considerations of a data warehouse. As revealed by Kimpel & Morris (2013), the three most impactful critical success factors to a data warehousing project are:

- Clearly defined business needs and benefits
- Quality of the source data
- Measurable business benefits

These three can and should be addressed during the conceptual modeling and when defining the project goals and scope. This will require collaboration and interaction with the customer, to be able to communicate in such away, that the mutual understanding is assured.

The model is drawn as a UML class diagram, which then will be exported as an XML-file, which SmartEngine imports and generates the object model to be used in the application. The actual Data Vault model is generated based solely on the Visual Paradigm export. This means that any changes will have to be modeled and then implemented. The data warehouse can never evolve without the conceptual model being updated.

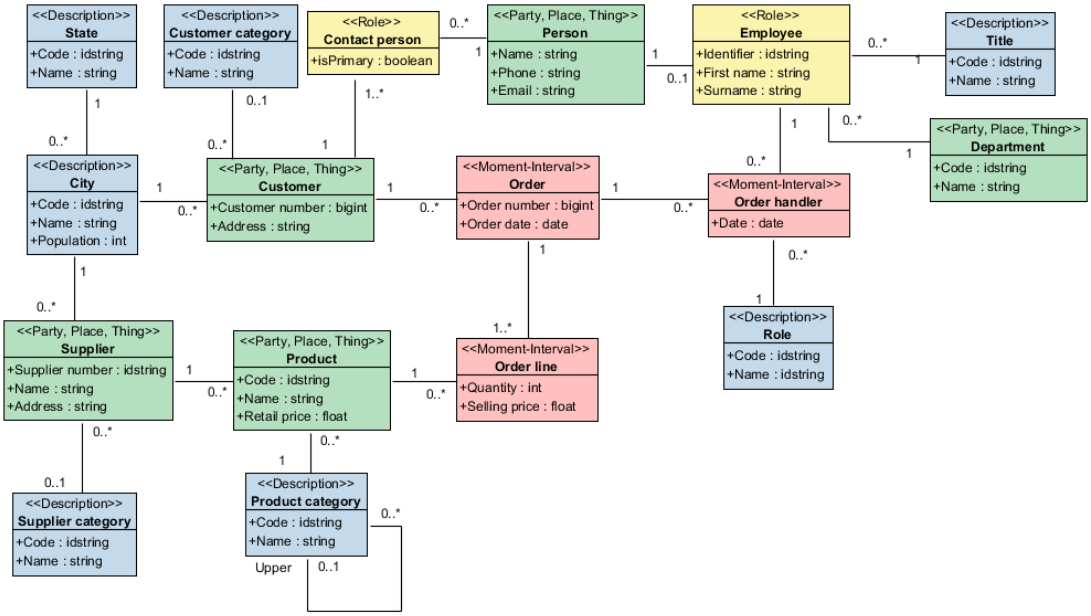


Figure 11 - Example of a conceptual model (Screenshot from Visual Paradigm, Eklund, 19.11.2017)

Figure 11 demonstrates how a model can look. The different colors of the entity describe the type of entity. Lines between entities depict a relationship. An entity can also have a line pointing to itself, depicting a parent-child hierarchy. The numbers at the ends depict multiplicity between entities. The text in the lower part of an entity lists its attributes, with the correct data type.

If two entities are connected by a line, it means they have a link in the data. It can be a direct referencing foreign key from the source system, or a reference that must be constructed using a combination of data to uniquely identify a row in its counterpart entity.

Multiplicity indicates how the objects relate in the depicted relationship. The number closest to the entity depicts the number of instances that are part of the relationship, of that entity. From Figure 11 we see that an order handler always has exactly one order, but an order can have zero or more order handlers. This will result in the generated “order handler”- object having a foreign key referencing to the “order”-object.

The most commonly used notations for multiplicity are:

- 0..1 – Zero, or exactly one instance
- 1 – Exactly one instance
- 0..* - Zero, or more instances
- 1..* - At least one, or more instances

Entities are modelled in different colors to describe the type of entity. Red stands for a Moment-Interval, which often means that the data have a validity period, or are time-reliant in some other way. An example is an employee handling an order at a given point of time. The same employee can handle the same order at two separate time periods. He might first be in the role a sales person, and later in customer service.

Yellow stands for Role. In Figure 11 we see that a person can be both a customer and an employee. The actual information related to the role is stored in that entity and, in most cases, placed between two entities to signify a many-to-many relationship. An example from Figure 11 revolves around the entities Customer, Person and Contact person. A person can be a contact person for several customers, and a customer can have several individuals as contact persons. This depicts a many-to-many multiplicity which is not possible to implement. As a result, a role-entity is placed in between. Then we have a contact person, who is a person, and is also connected to a certain customer. A customer can have several contact persons, and a person can be contact person for several customers.

Green entities are physical and tangible objects, such as party, place or thing. People, cost centers, products, departments are examples of green entities.

Blue entities are called descriptive entities. They are used for descriptive attributes, categorizations and classifications. Blue entities are usually the base for a dimension in the reporting layer. They are used to normalize data and remove redundancy. It is possible to use the same descriptive entity for several other entities. Looking at Figure 11, it would be possible to replace the three categorizing entities with a “Category”-entity, and a “Category type”-entity or “Categorization”-entity.

4.3.3 The source data

The data for the data warehouse comes from a variety of systems. In most cases the source is a transaction system, or a transaction database. Separate systems are used for different

purposes, such as financial data, HR data, Enterprise resource planning (ERP) and Customer Relationship Management (CRM). A DW is meant to combine the sources to provide a single truth, with limited redundancy, and without compromising the transaction systems' performance as discussed in chapter 2.1.1.

The source data are usually extracted directly from the transaction systems into flat files which then will be loaded into the staging area. An alternative is to utilize a direct connection to the operational database. With a direct connection, the data can be copied directly to the staging area, or to some sort of intermediary server's database. Either way, the load on the transaction system is not constant, but constrained to certain intervals and preferably at a time when the transaction load on the system is low. Usually this means during the night or during weekends. Because the data are stored at another location, the transaction system will continue to serve its purpose without additional load from the DW, as presented in chapter 2.1.1.

At this point all the sources are handled independently. The data are not modified between extraction and staging, and all validation of the data is done in the staging area. A successful and correct data extraction is vital when combining different sources.

Sometimes there is a need for manually managed data. Certain metadata or master data might be maintained in Excel-files or other similar formats. These need to be loaded, preferably automatically or through scheduled operations, and this places an emphasis on data validation. A manually maintained file must maintain the same structure as before, if it is to be loaded automatically. Manual files can easily turn into a high-maintenance issue unless precautions are taken, such as providing a static template for the files, or even a graphical user interface or an application, for data validation purposes. The goal is to have a process that everyone understands, to make sure changes are delivered as new source data, and to automate loading as much as possible. This will result in fewer problems, but also a faster response time for the end-user, as the data will be loaded without manual labor or intervention.

It is important to plan how the source data are to be handled. Will the data be extracted incrementally through subsets, or as a complete set of all the data? This impacts how the staging area is set up, and how the ETL-procedures are constructed. This will be examined further in the next chapter.

4.3.4 Staging area & ETL

The staging area is where the actual extraction, transformation and load will happen. The ETL is the most time-consuming part of implementing a DW when utilizing DWA, mostly due to tools like SmartEngine being able to create the database objects and the necessary procedures used to load them and, therefore, removing the manual labor involved in implementing the storage layer.

This is the phase in the work flow, where all the data will be cleaned up and transformed into a uniform format, suitable for the DW. Since the data originates from separate systems, which probably have been created by separate providers, it is very likely that data are stored in differing formats. For instance, dates can vary quite significantly from one system to another:

- Time zones – Some might be in GMT, while others are presented according to region-specific time zones.
- The separator between values
- The order of values
- Having leading zeros or not
- Whether time of day is included or not

All dates must be converted according to a certain standard to allow combination and comparison. Similar steps must be taken for other types of data that may occur presented in different formats when extracted from the source systems.

Redundancy should be avoided as much as possible and, therefore, data must be transformed and uniformed according to the ER model. An example of this can be found when looking at cost centers. It is highly likely that they exist in several separate systems. Both financial systems and HR-systems use cost centers. To eliminate redundancy and to allow comparison and aggregation, all versions of each cost center must be consolidated and the source data transformed, so that data from separate systems will point to the same cost center in the DW.

This is a very common scenario, and data are normally transformed and consolidated as described above.

SmartEngine

As presented in chapter 4.2, most of the database objects are generated by SmartEngine. One major part of the staging area is still done manually, the tables for the source data in its raw form. Both the tables storing the rows and the process of loading the data to the staging area, is mostly manual labor. A good practice is to create the tables completely according to the structure of the source data. The less transformation there is at this point, the easier it is to maintain and further develop the loading of the source data. This will be even more important when migrating towards Data Vault 2.0, where the raw data from each source system are loaded without any transformation and are combined later, in the business vaults.

Direct database to database

If the source data are copied directly from another database, it is normally done by configuring a linked server in SQL Server. This is the preferred way, because then direct SQL-queries from database to database can be used in stored procedures to initiate the extraction of source data. Not only is this a fast transfer, it is also easy to maintain and modify as necessary. Structural changes are easier to correct when the extraction is done with SQL directly from another database. The order of columns does not matter, compared to loading a CSV-file, when using SQL Server's bulk insert[3].

Flat files

An alternative way is to utilize flat files. This is a more common way of working, as the providers of operational systems might not want to grant access directly to the database for several different reasons. Usually an export of the source data is delivered by the provider, according to specifications and agreements. There is a large variation of specifications from system to system, due to reasons such as size of source database, frequency of export, number of exports for different purposes and separate customers. Sometimes the export is delivered as a specifically tailored database backup of the source database, containing only the relevant, and possibly limited, data.

[3] https://docs.microsoft.com/en-us/sql/t-sql/statements/bulk-insert-transact-sql

SSIS

The typical way of loading the source data to the staging area is by using SSIS. SSIS can handle a wide variety of sources and destinations, with support for error logging and error handling. SSIS-packages can be deployed to an Integration Service running on the SQL Server, which in turn allows scheduling and automatization of data integration jobs.

A common practice is to create a SSIS-solution for each source system or business domain, and have separate packages per filetype. Having the packages decentralized enables team collaboration and makes modification and further development easier.

With so many different scenarios and use cases, it is impossible to describe all possible data flows. Some of the aspects to consider when designing the data integration are:

- Is the source data complete, or incremental?
- Will all rows in the files be unique, or is it possible that modifications to existing and previously loaded rows appear in the files?
- What is the frequency of the exports, and how critical is it that the data are loaded as soon as it is accessible?
- Is the data in the files dependent on other exports?

Requirements and specifications such as these must be considered before starting to design the integration solutions. Usually the source data are imported once manually to allow for continued development of the staging area while also developing the integration, and because it is not certain that everything has been accounted for. Once the source table is finalized and the export deemed to be satisfactory and in its final format, the integration solution is finalized.

Regardless of how the source data are delivered or extracted, it is good practice to have an ETL-server, even when the direct database connections are available. Having a separate intermediary storage location removes dependency on the source system which is important, as presented in chapter 2.1.1. It is also a logical place to store and archive the files. Having everything in one place makes it efficient to back up. It is not uncommon that several data warehouses or data marts rely on common source systems. It saves time and resources to do the necessary pre-processing away from the data warehouses, in order to minimize the time it takes to load the data warehouse, once the last exports of source data have arrived.

The ETL-process and database objects generated by SmartEngine, have been presented in chapter 4.2.

4.3.5 The data warehouse

As presented in chapter 4.2, SmartEngine creates the database objects according to Data Vault theory, presented in chapter 2.2. Because everything is done based on the ER-model, any changes or additions to the data warehouse must first be modeled and after the model has been exported to SmartEngine, the changes and additions can be implemented to the data warehouse. In normal cases there is virtually zero manual SQL-labor involved when developing the raw storage layer of the data warehouse.

SmartEngine by default creates four databases for a data warehouse, one database each for the staging area, the ETL, the actual DW, and the DW-log. One of the reasons is to allow the use of separate hard drives or hard drive arrays for the databases, to maximize throughput. Another reason is to help manage the size of the databases. The process of backup and deployment is easier, and getting an overview of the DW is easier when it is split into several databases.

Updating the conceptual model

The conceptual model should be a continually evolving and changing model. As there is a heavy reliance on SmartEngine in all development, the conceptual model really is the driving force. This opens possibilities to interesting collaboration, as the business analyst can model new data to be added, and another developer can implement the change without detailed business knowledge. If the model is correct and technically feasible, the developer only needs to know where the new data are.

The heavy reliance on the ER-model has a disadvantage. If an entity is removed from the ER-model, SmartEngine effectively does not know about said entity anymore. There is no way for SmartEngine to be able to remove entities or objects from the data warehouse to match the ER-model. Therefore, it is vital to remove as little as possible from the ER-model.

Anything referring to an entity removed from the model, will continue to do so unless it is manually corrected in the database. A developer might encounter errors due to foreign keys, that were generated previously, but no longer are accounted for during SQL

generation, due to an entity being removed from the ER-model. These keys are not automatically removed and can lead to constraints that hinder the deployment of new entities.

Scheduling and monitoring

Scheduling and monitoring is managed through SQL Server Agent. The agent is a Windows service that runs on the SQL Server. The agent can execute various tasks, such as executing stored procedures, executing SQL-queries, executing SSIS packages, processing SSAS cubes, and much more. It has built-in error handling, logging, and can send e-mail through the database mail service.

The compilations of tasks are called jobs, and jobs can be scheduled on recurring schedules, to react to certain triggers, or as a follow-up to an alert or another job. This way certain jobs can trigger on the failure of another job, and other similar setups are possible. Several jobs can run simultaneously.

4.3.6 Semantic layer

The end-products of SmartEngine are the K-views and denormalized tables presented in chapter 4.2.6. Because they are simply manifestations of the ER-model, it is good practice to implement a semantic layer for the end-users. Both Kimball and Inmon propose the use of dimensionally modeled data marts, albeit at different phases of the data warehouse, as seen in chapters 2.1.2 and 2.1.3. The data marts are meant to be business-oriented and suit a specific need or requirement. According to Kimball, a data warehouse is simply a union of all data marts, whereas Inmon sees data marts as a subset of a data warehouse, isolated to suit a certain need, or to comply with various requirements on performance and security (Jukic, 2006).

Normally data marts are modeled as star schemas (Chaudhuri & Dayal, 1997). A star schema consists of a fact table referring to any number of dimension tables. An example of this can be found in Figure 3. A star schema is denormalized, having a fact table that should only contain measurements, metrics and values, with foreign keys referring to the dimensional tables with detailed descriptive attributes. The star schema is a suitable source for OLAP data cubes, as the denormalized tables mean there are fewer and simpler

joins to be made when querying and fetching the data. A star schema allows fast and versatile aggregations, which is vital in OLAP analysis.

An alternative to the star schema is the snowflake schema. It resembles a star schema, with a large difference being that dimensions are normalized to multiple related tables. The benefit of this is that redundancy is very low and, therefore, storage is efficient. However, the dimensional tables are considerably smaller than fact tables, and the saved storage space is not significant when looking at a larger scale. The disadvantage of requiring more joins than a star schema is so significant, that using snowflake schemas is not recommended by, among others, even Kimball himself (Kimball & Ross, 2013).

In our data flow, the data marts function solely as sources for OLAP cubes and, therefore, the star schema is a better choice due to fewer and simpler joins, resulting in faster data transfer.

Data cube

A OLAP cube can be described as a multidimensional dataset, designed for analysis of numeric measures. Examples of measures are sales, budget, revenue, and ROI. Each measure is influenced by the dimensions of the cube. The dimensions provide context for the measure and allow grouping and drilling. Examples of dimensions are time, product, and city or country. Depending on the choices in the dimensions, the numerical result of a measure will change. OLAP emphasizes the possibility of aggregation and comparison. A measure for sales will provide a different result, depending on the drilling and slicing of dimensions. The time dimension is particularly important, as it can be used for trend analysis and finding non-obvious patterns (Chaudhuri & Dayal, 1997).

The established practice for the company is to build tabular data models, using SSAS. Tabular is well suited for star schemas and supports the reporting tools used by the company and customers. Creating the data models is straight-forward, and tabular data models are very similar to the PowerPivot functionality in Excel. It is preferable to have several specialized cubes, instead of having one large cube containing everything. If supporting ad hoc-reporting is a requirement, the larger and general cubes are harder to understand, and the risk of misuse increases.

Normally cubes are specialized to a certain business area, or business domain, or an organizational unit. Security and data filtering is more manageable when the cube is specialized and limited to certain business areas.

Reports

Reporting in the context of business intelligence can be defined as collecting and presenting data to end-users, so they can be analyzed and function as decision support. It aims to turn data into insight in an understandable and efficient way. It also aims to bring various data sources into one consolidated view for the end-user. While Power BI has the capability to create an internal semantic model, it is more common to have ETL-procedures and a data warehouse or data mart, serving as the data source for reporting. It is possible, and not unprecedented, to skip the semantic layer and build a data model inside Power BI, when the data are simple, if the report is very specific and does not warrant the effort of creating a new data cube, or if the report is a one-time endeavor.

Almost all the company's customers rely on either Excel or Power BI as the reporting tool. Lately the trend has shifted more towards Power BI, due to its modern look and sophisticated visualizations. Power BI focuses on using graphs, plots and other similar visualizations to abstract, and clearly and efficiently convey information.

Power BI is designed as a self-service reporting tool, but can be used for enterprise-wide delivery of reports. Power BI and Power BI Desktop have been presented in chapter 4.3.1.

Some points to think about when planning and designing reports include:

- Who will use the report, and for what? Is it for analysis, or an overview of operations? Design the report for the audience and their needs.
- Prepare the reports for analysis. Maintain a consistent and understandable naming convention for columns and measures.
- Less is more, do not add every possible object into the report. Work with the customer and their analysts to find what they need. Remove clutter. Draw attention to the data, not the report or the visualizations.
- Present the data in a way that is easily interpreted, minimizing risk of misinterpretation. Use the appropriate visualization, not just the newest or fanciest. Try to tell a story.
- Design with privacy and security in mind. Take care when presenting sensitive data. Grant access to a limited group of individuals.

- Depending of the purpose of the report, you might want to show summarized data when opening the report, and allow drilling and deeper analysis when necessary and suitable.

4.4 Constructing the artifact

As presented in chapter 1.2, the aim is to continue with the established process of designing and implementing a data warehouse, with the caveat of everything being cloud-based. It is not a significant change, as we will be using the same modeling, the same tools, and the same tasks, but some aspects will be modified to suit a cloud environment.

The steps will contain the same tasks as with a traditional on-premise solution. For clarity, the construct will be divided into parts, according to the structure of chapter 4.3.

4.4.1 Tools and services required by the artifact

The following tools have been presented in chapter 4.3.1 and will be used in the same manner for the construct:

- Visual Paradigm
- SmartEngine
- SQL Server Management Studio
- SQL Server Analysis Server
- SQL Server Data Tools
- Excel
- PowerBI

Azure Storage

Azure Storage is a cloud service by Microsoft, that provides scalable, durable, and highly available cloud storage. Customers can store virtually limitless amounts of data, and pay according to usage. Azure Storage comes with options for local and geographic redundancy, and disaster recovery (Calder et al., 2011).

There are different storage types for different purposes. The most common ones are file storage and blob storage. File storage can function as a common network file share, and can be mounted and accessed via the Server Message Block (SMB) protocol. Blob storage is designed to accommodate larger amounts of unstructured data, such as binary data and media files.

Azure Data Factory

Azure Data Factory is a cloud-based data integration service, that is capable of orchestrating and monitoring data movement and data transformation. The emphasis is on extract and load (EL) rather than a traditional ETL-process. Data Factory comes with a variety of connectors for different SaaS and PaaS services, file shares, and other sources (Microsoft, 2018a).

Data Factory consists of pipelines and activities, which can be authored and managed in several different tools and APIs. The code is written in JavaScript Object Notation (JSON). Pipelines can be chained, so that one's output is another's input. This means that the second pipeline will start as soon as the first one has finished, assuming the scheduling is similar. For each activity, a source, a destination, and the type of activity performed on the data, must be configured. A pipeline can contain many activities (Microsoft, 2018a).

Azure Data Factory has a monitoring and management app for creating alerts that trigger according to rules. The alerts can be configured to send e-mail on certain types of events, such as a timeout or failure, while not reacting on a successful activity. Each alert and rule is attached to a certain pipeline (Microsoft, 2018b).

Azure SQL DB

Azure SQL DB is a relational database-as-a-service by Microsoft. It runs on the same SQL engine as an on-premises SQL Server, but usually gets new functions and enhancements before the on-premises SQL Server does. Azure SQL DB is meant to be a general purpose, dynamically scaling database for structured data. It is always running, guaranteed by a 99.99% SLA (Microsoft, 2017d).

Azure SQL databases can be scaled between the service tiers and performance levels manually or programmatically, without significant downtime. The service is billed by the hour, so it is fully feasible to scale up the database for intensive work, and then scale back down once finished.

Azure SQL databases are isolated from each other, and can be individually integrated with Azure Active Directories. Encryption and auditing capabilities are built-in. The querying language is T-SQL as with the on-premises variant, but there are some

differences. Queries across databases are not supported, unless the databases are in an elastic pool.

Elastic pools can be created, where several databases are placed in a shared resource pool. For unpredictable and irregular performance demands, having a larger shared pool to draw resources from, allows dynamic scaling without intervention. Minimum and maximum limits can be defined per database, and the pool resource tier can be scaled up and down.

Azure SQL DB scales up to a maximum size of 1 TB. Some limited regions offer a higher performance tier that allows up to 4 TB maximum storage per database. Microsoft also provides a service meant for EDWs, Azure SQL DW. The pricing is very different, as Azure SQL DW utilizes a Massively Parallel Processing (MPP) architecture, meant for running complex and large queries of up to petabytes of data. The Azure SQL DW can be paused and resumed within minutes, when loading or querying is needed.

Azure SQL DB is designed for OLTP, while Azure SQL DW is strictly for OLAP. The SQL DB can also be used as a platform for a DW, with the maximum database size being a clear limitation. As presented in Table 22, the Azure SQL DW has a limited capacity for concurrent queries and connections, and often it is beneficial to have one or several data marts (Azure SQL DB) as the data source for ad hoc querying and reporting.

Feature	SQL Database	SQL Data Warehouse
Size	Max of 1TB for a database	No limit
Concurrent Queries	Up to 6 400	Up to 32
Active Connections	Up to 32 000	Up to 1 024
Cross-database Queries	Supported	Not supported
Pause/Resume	Not supported	On-demand pause/resume of resources
Scalability	11 tiers	12 options
Replication	Supported through secondary databases in different regions (up to 4)	Not supported
In-Memory OLTP Tables	Supported	Not supported
Polybase	Not supported	Supported

Table 22- Comparison of features of SQL DB and SQL DW (https://key2consulting.com/comparison-of-azure-sql-database-to-azure-sql-data-warehouse/)

4.4.2 ER modeling

The business case revolves around modular packages. To support these packages, and allow development that benefits new and existing customers, a shared ER-model will be designed. The product will have one single ER-model, used by all the customers. It will be constantly evolving as new features are added and as modification is necessary, to suit a new customer.

No changes to the modeling tool will be necessary, as the ER-model is exported to a XML-file, which will be imported into SmartEngine. The process of modeling will continue as described in chapter 4.3.2.

It is vital to consider old implementations when making changes to the model, as all development work in SmartEngine is based on the ER-model. As previously mentioned, removing or renaming entities should be avoided to minimize the labor required to keep old implementations up to date. Maintaining a generalized and versatile ER-model becomes even more important, as several customers with multiple different source systems and source data rely on the same model.

Nothing changes with the conceptual modeling. The modeling techniques and types of entities remain the same. The addition of some metadata to the ER-model is required, because SmartEngine will need to know what platform the ER-model is being implemented on. To keep the customers separate, metadata will be added to keep track of which entities a certain implementation uses. This entails that not all new entities are automatically implemented in every data warehouse instance. This will be done according to needs and business decisions. However, the aim is to make it easy for customers to expand their data warehouse and add new features that might have been developed for someone else. From the company's point of view, after one initial development cycle, implementing the same feature for other customers should be faster, easier, and be of higher quality.

4.4.3 The source data

Exports as flat files

The source data will be provided as flat files, extracted from the source systems. The customers' system providers will manage the extraction and delivery of the files to the chosen storage solution. Files are categorized by a strict naming convention, with timestamps to make it easier to determine the correct (and newest) export meant to be loaded. The data will be provided in CSV files.

To make it easier for the providers, the construct will rely on an incremental approach to the source data. The size of financial and HR-data is constantly growing as time passes. However, it is uncommon to make changes to historical data of this kind. Therefore, there is no need to keep extracting and loading all the data. It is more suitable to choose a time period, in which it is likely that change will occur.

The structure of the exports will be specified according to previous experience. Especially when considering financial data and HR data, there is a limited number of systems in use on the market. The odds of having worked with said systems are high. Reflecting on these projects and drawing from experience, will help in specifying the correct format and content of the exports.

Because the ER-model already exists, it may be beneficial to strive towards a model driven specification for the source data exports. If necessary, the ER-model can be expanded, in case the source system contains data that have not yet been modeled.

Azure Storage

The storage solution type chosen for this construct is binary large object (Blob) storage. Blob storage can contain various file types, such as images, video, text files, and Excel files. The source data will be separated per customer, and per business domain or source system. It is crucial that there is zero risk of customers' source data being mixed. It is assumed that customers will not be granted access to the exports of the source data, but to ensure privacy, separate storage accounts per customer will be used. There should be no need for customers to gain access to the exports, as they already can examine the data directly in the operational system.

The files will be organized in a hierarchy of virtual folders. This allows the ETL to be built to handle all files in any given folder. Ideally, the files should be archived away from the main folder, once they have been loaded successfully. This will be done to ensure that any file is loaded only once, to save execution time and lower redundancy.

Azure Storage is a good choice for this construct, as it is a platform provided by Microsoft, meaning the integration with other Microsoft tools and platforms is solid. The ability to scale dynamically and pay according to usage results in lower operating costs, and easier expansion in the future.

4.4.4 Staging Area & ETL

Normally SSIS is used as the primary ETL-tool. Because this is going to a fully cloud-based solution, the only option would be to allocate a virtual machine in Azure, with the necessary SQL Server license to allow the use of SSIS. Having a separate server, which only exists to enable the use of SSIS is expensive and unnecessary. Therefore, Azure Data Factory will function as the ETL-tool. In case Data Factory is not sufficient, the virtual machine will be provisioned, and this server will be used for all customers' implementations, as the ETL-platform.

As discussed earlier in this chapter, the source data will be delivered by the source system providers in a specified format, at a specified frequency, and to a specified folder in Azure

Storage. Each source folder will have its own pipeline. The scheduling will be configured according to the estimated time of arrival of the source file. The pipeline will examine the contents of its designated source folder in Azure Storage and copy the contents of every file to its designated table in the staging area. A stored procedure will be used, that only inserts the new rows to the table. Any existing rows are ignored. Once the content has been transferred, the file will be archived. This will be repeated for each file in the source folder. No transformation will be done during the data movement.

Azure Data Factory has built-in capability for scheduling. However, the scheduling is rather limited, as it can only utilize schedules based on time. There is no functionality for triggered events like having pipelines run when a file appears in Azure Storage. However, because the frequency and schedule is specified, Data Factory can be relied on to handle data movement to a satisfactory level.

There will be a final pipeline that starts the loading process of the DW, which is presented in chapter 4.2.7. This pipeline will start when every data movement activity has finished. This is to ensure that the DW is loaded with consistent data, as there will be dependencies between source systems. Discussion with the customer is necessary to establish the critical systems, which are required for a DW load to occur. Not all source systems are critical.

SmartEngine

As presented in chapter 4.3.5, SmartEngine creates four databases by default. Azure SQL DB has some technical limitations, mostly due to databases being isolated, even though a virtual Azure SQL Server exists. For instance, no cross-database queries are possible without an elastic pool, and no global temporary databases exists.

These two limitations require modifications to SmartEngine. As presented in chapter 4.2.2, SmartEngine relies on global temporary tables when loading the DW. Therefore, SmartEngine will be modified to create a physical ETL-table for every entity. The ETL-table stores both the original data and the converted data. This helps troubleshooting, in case some data are not loaded properly. The entity's OUT-table is used as source for the ETL-table.

The other change is to create a single database for each implementation, instead of the usual four databases. In practice this has low impact, the separate parts of the DW will be distinguished through the use of schemas.

Aside from these changes, most of the work involving the staging area will occur as described in chapter 4.3.4.

4.4.5 The data warehouse

The storage layer of the DW will be very similar to what is described in chapter 4.3.5. Azure SQL DB and on-premises SQL Server share the same SQL engine and, as such, the established way of implementing the storage layer and loading it is unchanged.

One infrastructural difference, is having the possibility to choose the performance level. Because Azure SQL DB is a scalable service, there is a clear benefit of being able to increase the performance for any computing-intensive workloads or data transfer workloads, and decreasing it once the workloads have finished. The scaling can be done through the Azure portal, PowerShell, the REST API, or even using T-SQL. Scaling the database up when starting the load batch, and scaling back down once finished would allow a major throughput increase when needed, and result in cost savings when the DW is idling.

Azure SQL DB is deemed to be sufficient for this type of construct. As it is not an EDW, but a compilation of smaller modules, going for the Azure SQL DW is excessive. If the project is successful and customers start adding up, consideration must be made to migrate towards an elastic database pool, to better serve the needs of the customer, and to cut down on operational costs.

Scheduling and monitoring

Azure SQL DB does not provide a SQL Server Agent, that usually is used to orchestrate and monitor all aspects of the DW. For this construct, Data Factory will be used. The data transfer is already handled by Data Factory and, therefore, adding the activities to start the preparation and loading of the DW, along with refreshing the data marts and SSAS cubes, fits nicely into the process.

4.4.6 Semantic layer

Internal data marts will be created as per the established way of working, presented in chapter 4.3.6. Star schemas consisting of facts and dimensions will be created to serve as data sources for the SSAS cubes.

Azure has not yet released SSAS as a cloud service, which means that the best workaround is to provision a virtual machine in Azure, with SQL Server pre-installed. The SSAS data models will be authored on the virtual machine, using SSDT. An alternative would be to build the data models in Excel reports, using Power Pivot. However, the Power Pivot data models cannot be accessed outside the report, meaning every report must have its own data model, which quickly gets cumbersome to maintain. Having the data models separate from the Excel reports better cater to the advanced users, who want to create their own reports from the SSAS cubes.

All customers' cubes will reside on the same server for cost savings. The performance of the virtual machine can be scaled up and down, as almost every Azure service. A lower performance tier can be used during development, and the machine can be scaled up when encountering performance issues.

Access to the cubes will be restricted per customer. Separate user accounts will be created for each customer, so no customer has access to another customer's DW, or cubes. Two separate accounts will be created per customer, one that has access to the DW and will retrieve the data from the DW to the SSAS cubes, and a second account that has read access to the data cubes in SSAS. This account will be used when consuming the reports.

Having one account for each customer's end users entails that row-level security cannot be utilized. This will be implemented as soon as Azure Active Directory integration is possible, most likely in combination with SSAS as an Azure service, when it is released.

To ensure that access to the cubes is limited, only certain IPs will be allowed to access the SSAS data cubes. There is a risk of the Excel reports getting into the wrong hands, but the firewall will limit the access to specified IPs.

The primary reporting tool will be Excel. Excel is a valid choice, as it is very likely already in use in any organization, and hence will not result in added costs. A survey among part-time students at Newcastle Business School, conducted by Pemberton &

Robson, found that only 13% of respondents did not use spreadsheets in their work. They additionally found that 94% of spreadsheet users relied on Excel (Pemberton & Robson, 2000).

Most of the frequently used add-ins for business intelligence are free of charge. The Excel connectors allow a direct connection to the virtual machine and the SSAS cubes. The report will consist of pivot tables, graphs, and charts. The aim is to keep all the calculations and measures in SSAS, that way any modifications or improvements will benefit all reports that rely on the cube.

The reports will be delivered to the IT-team of each customer, who then distributes them to the end users. This will most likely be through a network share, or SharePoint site.

An alternative that is considered, is using Power BI. Power BI is not free, as every user needs a Power BI license linked to their organizational account. Using Power BI requires more integration with the customers' domains. Power BI requires organizational accounts, it will not work with personal e-mails. Because the same server holds the cubes of all customers, it cannot be merged into one domain. This leads to a rather complicated architecture of establishing domain trusts between multiple domains and the virtual machine. This might not be acceptable by the IT-administrators of the customers.

One way to overcome this, is to publish the data with the Power BI reports, and refresh it on a schedule. That way the cubes can continue to serve several customers and still reside on one single server. Using the import method means that security and role management must be done on report-level, instead of source-level.

Once SSAS is provided as an Azure service, this will be re-evaluated, but in the first iteration, Excel will be the reporting tool.

4.5 The implementation

Implementation of the artifact began in December 2015 and was designed according to the services and tools available at the time. Some things changed, or were added, during 2016 that were not originally part of the construct. Looking back, several things could and should be improved. The first iteration of the implementation was done in 2017.

The most difficult challenges related to authentication, identification, and delivery and consumption of reports. Also, the tool used for orchestrating the ETL-process had severe limitations, which will be presented in chapter 4.6.1.

The implementation of the solution spanned over a long period of time. Due to this, the availability of newer and better tools and services caused some changes in the architecture, especially in the semantic layer.

4.5.1 The technical architecture of the first version

The source data are stored as comma separated values (CSV) files, in Azure Storage. Every data warehouse has its own container and the CSV files are grouped in virtual folders according to business domains. The service company we are collaborating with pushes the source data into the Azure Storage containers. We do not have access to any data outside of what is found in Azure Storage.

The ER modeling and implementing of the ETL and data warehouse was done according to the artifact. Johnsson implemented the minor changes in SmartEngine, presented in chapter 4.4.5. The workflow of creating a data warehouse in the cloud was very similar to the previous experience of working on on-premises solutions.

Data Factory is used to extract and load the source data to the staging area. No transformation is done during the extraction, but is instead handled in the staging area. This is mostly due to limitations of Data Factory, examined in chapter 4.6.1. The new source data are loaded every night to the staging area, and the data warehouse is also loaded every night, once the source data transfer is done. All the source data are gathered in the staging area, so a full load can be done at any time, if necessary.

The hubs, satellites and links are generated as presented in chapter 4.2. Azure SQL DB and SQL Server run on the same database engine and, because of that, very little tweaking of SmartEngine and the previous workflow was necessary.

For the semantic layer for the first version of the solution, we decided to provision a virtual machine in Azure, with SQL Server and SSAS installed. The SSAS cubes were authored on that virtual machine. This setup proved to be challenging. It meant only two people could work on the data models at a time. One requirement for the solution was to enable single sign-on for the end-users, which was not possible with this setup. End-users

had a shared customer-level account that was used to gain access. This also ruled out row-level security, as we could not identify the end-user.

We decided to use this setup, to continue development, and wait for the Azure Analysis Services (AAS) we knew was close to release. We opened the firewall on the virtual machine to certain IP-addresses to allow Excel to connect to the SSAS data models. The Excel-reports, and any changes to them, were delivered by e-mail to the customers' IT-teams, who distributed them internally to the end-users. During early development, there were several versions of the reports circulating, with no certain way of knowing whether it was the newest version or not. At a later stage, the reports were embedded to the customer's SharePoint-site, which solved this problem.

The first version of the solution was not satisfactory to the development team, the service company, or the end-users. Too many requirements were unfulfilled and the solution was not reliable.

4.5.2 The technical architecture of the second version

We made a significant architectural restructuring once AAS was released. We shut down the virtual machine, and adopted AAS. The data models were easily migrated to AAS, which removed the limit of only having two developers signed-in at a time. The data models could now be authored from our personal computers. The Visual Studio projects were maintained in Git to ensure that everyone worked on the most recent version, and did not overwrite each other's changes.

This switch solved the issue of row-level security and identifying the end-user. User accounts were created directly into the AAD, which meant we could implement row-level security as we could identify the end-user. Single sign-on was not possible yet, which caused some frustration on the customer's part. We learned of a feature called B2B, which was going to be added to the AAD. It would enable tenants to invite external users, and grant access to resources and tools, as if they were internal users. This feature arrived six months later than promised, which again added friction between the customer and the development team. Eventually, it was released and we could finally start working on granting access to the end-users on their own organizational accounts.

Due to limitations in Data Factory, presented in chapter 4.6.1, we had to process the SSAS data models using Azure Automation. Without a way to trigger the processing based on successfully completed Data Factory activities, we had to define schedules, and have the SSAS data models process at certain times, regardless of potentially failed DW loads. More than once, this caused the data models to process even when the building of the Fact-tables had failed, and this resulted in reports with missing data.

Simultaneously, we also switched from Excel to Power BI as the reporting tool. Power BI works flawlessly with AAS, and was a clear choice by both the end-users and the service company, due to its modern visuals and look of a more modern tool, compared to Excel. Even though Excel is a widely used tool in organizations (Pemberton & Robson, 2000), customers seem to view Excel as archaic and ill-suited for reporting.

The switch over to Power BI was relatively simple, as the underlying data cubes already existed with all the facts, dimensions, and measures we needed, and we just had to design the reports and implement them. No additional architectural re-design was necessary.

Migrating to Power BI also solved the problem with distribution, and maintaining versions. The reports are accessed through the online Power BI-portal, which ensures users are always accessing the correct and newest versions of the reports. In the future, the reports will be embedded into SharePoint Online, to further integrate the reports with other organizational material.

4.6 Examination of the artifact against the implementation

In this chapter some new technologies and services will be discussed, that have been introduced into the solution since the pilot project. Some issues related to the ETL-tool chosen for the artifact will also be presented.

4.6.1 Shortcomings of Data Factory

Missing features and functionality

We encountered several issues when working with Azure Data Factory as our data integration tool. At first glance it seemed like a simple and robust serverless service,

however, we faced severe challenges with the transformation capabilities, scheduling, and lack of support for any marginally complex activities.

Data Factory is marketed as a hybrid integration service, combining both an ETL- and an ELT-tool. However, we were limited to copying content from CSV-files to our staging area, without the capability of archiving loaded CSV files, or ignoring already loaded CSV files. This resulted in us either loading every file in the folder, every time, or alternatively designing a logic that would find one file based on the filename, and the date when the activity ran. In most cases this would be satisfactory, but if the file arrives a day late, it will never be loaded into the DW. We chose to load all files every time and manually archive the files at certain intervals. We gathered all history data in the staging area, and using SQL, we selected the correct subset that resulted in the most complete set of data.

Data Factory lacks a built-in support for processing the SSAS data models. It is possible to develop custom activities in C# and deploy those for use in Data Factory pipelines, but without the knowledge of C#-development, even the smallest task turned out to be time-consuming and difficult.

Data Factory is a very strict tool, meaning that everything about the activity and the datasets must be defined and declared. If a column is missing, or there is an unexpected line-break or escape character somewhere, the whole processing fails. The order of columns must remain the same between files, even though every source column and destination column are mapped pairwise in the UI.

Scheduling

The scheduling mechanism in Data Factory is very awkward. It is based on slices; the datasets are sliced in certain intervals. Intervals can be defined in minutes, hours, days, weeks, or months. We could not understand why scheduling was implemented in this way. It is difficult to understand, and we have not found any advantages so far.

Data Factory relies on input datasets, output datasets, and activities that operate on these datasets. All three types must have identical slicing intervals and start times. As a result, if you want to change the time for when a run starts, or the frequency of execution, you must change parameters on all three types.

Effort to maintain and develop further

There is no built-in versioning, and no way of attaching a third-party version control system. There also is no easy way of testing a change to the Data Factory objects. Changes must be deployed, either on top of existing code when feasible, or by creating a parallel pipeline and deleting the previous after successful testing. As soon as the change is deployed, the pipeline starts running all slices that fit between the start time and the end time, defined for the pipeline. To avoid this, the starting time for the pipeline must be configured to a future point in time. These small issues pile up, and make the use of the tool very awkward.

4.6.2 New tools and services

4.6.2.1 Azure Analysis Services

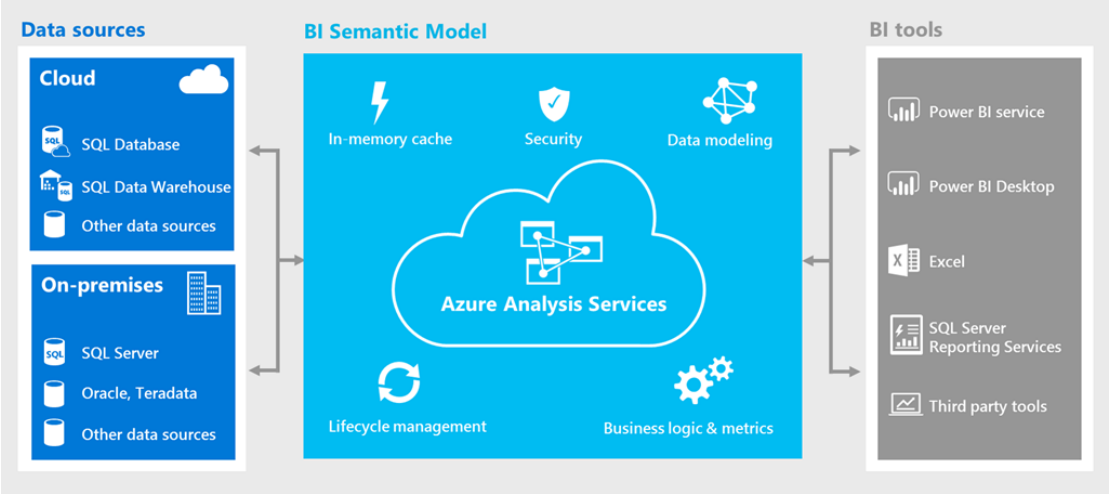


Figure 12 - Azure AS architecture (https://azure.microsoft.com/en-us/blog/announcing-azure-analysis-services-general-availability/)

Azure Analysis Services became generally available 19.4.2017. Azure Analysis Services is a PaaS OLAP engine, based entirely on the SQL Server Analysis Services. It supports a hybrid approach of using both cloud and on-premises data, or only one of them. Being an Azure service, means it is well integrated with the Azure data platform.

Azure Analysis Services relies on AAD, and enables Single Sign-On (SSO) by inviting organizational users to the tenant AAD and granting access to specific data models. This solves one of the problems presented in the construct regarding authentication.

Data models are authored using SSDT and Visual Studio. There is practically zero difference between Azure Analysis Services and SQL Server Analysis Services from a developmental point of view. All the same functionality and capability is present in the Azure Analysis Services.

Azure Analysis Services was launched after development began on the first solution, and not having this service was the biggest obstacle in creating a fully functioning solution for the customer. Authentication, row-level security, and keeping customers' data models separate from each other became non-issues, once Azure Analysis Services became available. Azure Analysis Services offers good support for both Power BI and Excel reports.

4.6.2.2 Azure Active Directory B2B

Managing access rights and authentication became non-issues, once Azure AD business-to-business (B2B) was launched. General availability of AAD B2B was announced 12.4.2017. B2B enables organizations to invite external users into their own organizational tenant as guests, whether the invited users' organizations utilize an Azure AD or not. The invited users can be given access to data, resources and applications, as if they were part of the inviting organization. However, the users navigate and consume resources using their own credentials and, therefore, there is no username and password redundancy, and everything related to the user's credentials is managed in the user's home organization.

4.6.2.3 Data Factory v2

Data Factory v2 was announced at Ignite, in September 2017. V2 will be separate from v1 which we used, and there will be no compatibility between them. In practice, it is a completely new tool. V1 will be supported, but will not be developed further. V2 brings several new and missed features, that improve the capabilities of Data Factory to function as a fully-fledged ETL and integration tool.

With v2 comes the ability to deploy and run SSIS packages in Azure in a dedicated environment called Integration Runtime (IR). The packages will be developed in SSDT, and deployed to the IR. The SSIS running on an IR is very close to a traditional on-premises SSIS presented in chapter 4.3.1, except for the inability to utilize third party addons and extensions in SSIS. Having all the SSIS capability is a considerable advantage

in development, as SSIS has significantly better features and transformation capabilities. Data Factory v1 was essentially limited to copying data from one location to another.

The scheduling has been revamped, and expanded. The slices from v1 are gone, and scheduling can now be done by a simpler and more intuitive time-based definition. There is also support for using triggers to start jobs and dataflows. The triggers can come from inside Data Factory, through the REST API, or through web hooks via Azure Automation or Azure Logic Apps.

Authoring Data Factory pipelines and packages can now be done through Azure Data Factory V2 Visual Tools, a graphical UI in Azure based on drag & drop. Parameters and metadata are configured through this UI, and it is no longer necessary to implement changes through writing and rewriting JSON.

5 DISCUSSION AND CONCLUSION

The process of developing a cloud-based data warehouse when relying on SmartEngine, is very similar to an on-premises solution. The established strategy is mostly unchanged. The changes to SmartEngine were easy to implement. Because SmartEngine is designed to implement models, instead of being a strict automation tool, there is a flexibility that cannot be found in competing commercial tools.

Some technical details vary between cloud and on-premises environments, but Microsoft has clearly invested time and effort into Azure, to enable an easy and effortless migration for those looking to move to the cloud, but also provide a diverse offering for anyone looking to start building from a clean slate.

With specific tools and services such as the Azure SQL DW[4] being offered for Enterprise Data Warehouses, the argument by Baars and Kemper (2010) in chapter 2.3.4 about cloud-based data warehouses not being possible or a wise choice, seems to be a partially valid, yet outdated concern.

The challenges of the project were mostly technical. Presumptions and misunderstandings led to some issues when integrating and combining all the services. The issues could mostly be attributed to the project being a pilot project and experimental by nature. A more thorough planning and deeper familiarization with the tools and services could have prevented some of the setbacks. We recognized that a few tools and services were missing when the project started, but we could not wait for them to be released, as no release dates had yet been announced.

As the business case revolves around delivering easily reproducible modular packages, a cloud platform seems to be a good candidate, due to the quick and easy way of provisioning new resources in the cloud. As the data warehouses start small and eventually expand, the scalability provided by PaaS-providers such as Microsoft Azure supports the increasing size and performance requirements. A data warehouse is mostly in use during the loading and when populating the semantic layer. On a cloud platform,

[4] https://azure.microsoft.com/en-us/services/sql-data-warehouse/

it is possible to scale down during the downtime and scale back up in anticipation of the loading cycle.

5.1 Purpose and effect of SmartEngine

Development of SmartEngine began mostly due to the lack of any existing commercial tools. The idea behind SmartEngine also differs from modern DWA-tools in that is a tool for implementing models. It simply replaces part of the work that normally would be done. Very little is done using the SmartEngine UI, other than generation of the necessary SQL-code, which is run using SSMS.

Due to its design, it would be easy to extend the automation capabilities. Currently, the structure and loading process is generated in SmartEngine. Logical next steps would be to model the incoming source data and generate the staging area, or to model the facts and dimensions and have SmartEngine generate them as well. It could even be possible to model the whole semantic layer, and have SmartEngine generate the code for implementing the data models in SSAS. It is all about finding the patterns and rules for generating a general object, and then modeling it.

SmartEngine is vital to the company's daily work. Any data warehouse, for any customer, will end up with a similar structure and architecture. It enables developers to switch between projects quite effortlessly, because the working process does not change between projects. There is a high level of standardization and uniformity.

5.2 General applicability of the artifact

The construct of this thesis is limited to Microsoft tools and services. Based on the diverse offering of tools and services in Microsoft Azure, it is safe to say that the prerequisites for migrating a Microsoft-based environment, from on-premises to the cloud, are perfectly met.

Just as on-premises work is divided into parts, with specific tools for each task, so too are specific tools and services for specific needs offered in PaaS-environments. It comes down to finding the correct services and preparing for the technical differences in advance.

The artifact constructed in this thesis is reliant on SmartEngine. However, SmartEngine is merely used to implement and maintain the data vault entities in the database. The Azure SQL DB is so similar to on-premises SQL Server that any tool or process that works with SQL Server should work with Azure SQL DB, after some slight modifications.

5.3 Future research

How does Microsoft Azure compare to other PaaS-vendors?

This thesis was limited to Microsoft platforms and tools. It would be beneficial to investigate other vendors and their offering of tools and services. There might be differences in prices and offerings, as well as the added benefit of being able to choose a provider that the customer already uses and is familiar with.

Are there vendors that provide one complete package, instead of a decentralized offering of PaaS-modules?

One central aspect of Microsoft Azure is the decentralization of services and modules. On one hand, this allows choice and combination of the preferred services and modules and to start using them when needed but, However, there is no single service that delivers the complete package. It could be beneficial to investigate whether this kind of packaged solution exists, where everything is integrated and combined as a full-fledged service.

What is the most efficient way to deliver new iterations of a previously implemented module to other customers?

So far, only a first modular solution has been delivered. Moving forward, it would be wise to look for existing best practices, or to come up with one's own strategy for efficiently delivering further iterations of the solution for future customers. The base offering remains the same, but there should be research done on how the existing implementations could be benefited from, when delivering the same module to new customers. Not all customers have the same source systems, or even use the systems in the same manner.

How would the source data or the semantic layer be modelled for automation by SmartEngine?

SmartEngine has proved its worth in the day-to-day development cycle. What would it take to expand the capabilities towards either modelling the source data, for automating the staging area, or towards the semantic layer by modelling facts and dimensions? How would all the business logic and rules be modelled to automate the creation of measures and other aggregation?

6 SVENSK SAMMANFATTNING

Bakgrund

För att ledningen skall kunna fatta informerade beslut, krävs det stödande data. Oftast finns all data tillgänglig internt i organisationen, men det krävs insats för att omvandla data till information. För detta ändamål används ofta informationslager (engelska *data warehouse*).

Enligt Kimball (2013) år målet med informationslager att upprätthålla en konsistent och korrekt samling av data från flera operativa källor. Eftersom de olika källorna inte nödvändigtvis är fullständigt integrerade, leder det till att det kan finnas många versioner av samma affärsobjekt. Ett informationslager skall kombinera dessa, så att data från flera källor är direkt jämförbara. Informationslagret måste vara bästa källan för information, annars kommer den inte att accepteras och tas i bruk (Watson et al., 2002).

Syfte

Skribenten jobbar för ett företag, som producerar och levererar rapporterings- och analyslösningar. Företaget utnyttjar SmartEngine i skapandet av informationslager. SmartEngine är ett verktyg som utvecklats internt, som automatiserar implementeringen av databastabellerna, samt laddningsprocedurerna i informationslagret, genom att generera Structured Query Language (SQL)-kod. Syftet med avhandlingen är att undersöka hur processer och rutiner måste ändras för att kunna utnyttja molntjänster, exempelvis platform as a service (PaaS). I samband med det undersöks vilka verktyg som måste ersättas med en variant utvecklad för molntjänster.

Uppbyggnad och forskningsmetoder

Avhandlingen består av två delar. Den första delen är en litteraturöversikt, där de centrala teorierna kring informationslager, molntjänster och automation undersöks. Den empiriska delen består av en kombination av konstruktiv forskning och en fallstudie. För att det skall levereras en kundlösning som bygger på molntjänster, kommer en artefakt att konstrueras, som baserar sig på vetenskaplig litteratur och tidigare projekterfarenhet. Ifall artefakten visar sig vara implementerbar, kommer den att fungera som plan för projektarbetet. Projektet där lösningen implementeras kommer att analyseras i form av en

fallstudie. Skillnader mellan artefakten och den slutliga lösningen kommer att presenteras, samt observationer under projektets lopp.

En väsentlig del av konstruktiv forskning är att analysera ett problem, hitta en lösning som grundar sig på teori, och undersöka ifall lösningen går att utnyttja på en generell nivå. Även om det handlar om att lösa ett specifikt problem, så är det sannolikt att lösningen går att tillämpa i viss mån, på andra liknande problem (Kasanen, Lukka & Siitonen 1993).

Enligt Kasanen et al. (1993) har konstruktiv forskning flera särdrag. Forskningen sker stegvis och i faser. Processen är klart uppdelad, och varje del går att granska separat. För att konstruktiv forskning i grund och botten handlar om problemlösning, är konstruktionen av artefakten en väldigt målinriktad aktivitet. För att avhandlingens mål är att lösa ett riktigt problem, passar konstruktiv forskning väldigt bra som forskningsmetod.

För att problemet som skall lösas är ett verkligt fall, är det naturligt att uppfölja projektet som en fallstudie. Vanligtvis används fallstudier då problemet undersöks i en naturlig och verklig omgivning. Fenomen kan med framgång forskas i via en fallstudie, ifall den vetenskapliga litteraturen kring ämnet är begränsad (Benbasat et al., 1987).

Benbasat et al. (1987) konstaterar vidare att vanligtvis ligger fokuset för fallstudier på implementationen, samt orsakerna till varför ett fall lyckats eller misslyckats. Målet eller hypotesen är sällan definierad på förhand, och fallstudier är ofta explorativa till naturen. Denna syn på fallstudier passar väl in med problemställningen i avhandlingen, och därför analyseras implementationsprojektet som en fallstudie.

Teori

Det finns två etablerade definitioner på vad ett informationslager är. Den första är: en subjektorienterad, tidsvarierande, konstant och integrerad samling av operativa data, som stöder ledningen i beslutfattningsprocessen (Inmon, 2005). Den andra definitionen är av Kimball (2011): ett informationslager är en kopia av transaktionsdata, sammanställd så att den stöder sökningar (engelska *queries*) och analys.

I skribentens företag modelleras informationslager som datavalv (engelska *Data Vault*). Datavalvets arkitektur består av ett nätverk av hubbar, länkar och satelliter. Hubbar används för att identifiera affärsobjekt (engelska *business object*). Identifiering sker via

en affärsnyckel (engelska *business key*) Hubbarna innehåller metadata om när objektet har laddats. Länkar modelleras som ett förhållande mellan två hubbar. Länkar kopplar samman två affärsobjekt. Satelliterna innehåller beskrivande data för varje entitet. Grundtanken med ett datavalv är att ingenting raderas. Ändrade data läggs till i databastabellerna, och via metadata upprätthåller man tidsperioderna för när en viss version av data gällde (Linstedt & Olschimke, 2015).

Projekt där informationslager planeras och implementeras är riskfyllda. Det uppskattas att 40–50% av projekten misslyckas. Det är väsentligt att ha väldefinierade behov och krav för projektet, en god förståelse av sin data, samt upprätthålla hög kvalitet på data i de operativa systemen (Kimpel & Morris, 2013; Sen et al., 2012).

För att minimera risker, förkorta projekt, samt minska på kostnader, har det utvecklats verktyg som automatiserar delar av implementeringen av informationslager. Historiskt sett har automation använts för att förbättra kvalitet, öka på effektiviteten, minska på manuellt arbete samt förkorta utvecklingstiden (Eckerson 2015b). Då automatiseringen bygger på mönster och regler, är det möjligt att ändra på sin design och implementera ändringen, utan att mista det som tidigare gjorts. Utvecklingen blir mer smidig och planer kan ändra på kort varsel (Rossi, 2015).

Molntjänster har blivit lockande inom business intelligence. Enligt National Institute of Standards and Technology kan molntjänster definieras som en samling delade resurser och delad hårdvara som kunder och användare snabbt kan ta i bruk och ta ur bruk enligt behov, utan att behöva bry sig om uppehåll (Mell, 2011). Löften om lägre operativa kostnader, hög flexibilitet och möjlighet att skala resurser enligt behov är de viktigaste drivkrafter för utnyttjandet av molntjänster (Agrawal et al., 2011; Baars & Kemper, 2010; Marston et al., 2011).

Informationslager i molnet är ett verkligt alternativ, speciellt för små och medelstora företag. Med molntjänst skulle det vara möjligt att välja prestanda enligt behov, och betala enligt användning. Att kunna ta tjänster i bruk utan att hamna investera i förväg underlättar utvecklingsarbetet (Krishnan, 2013).

SmartEngine

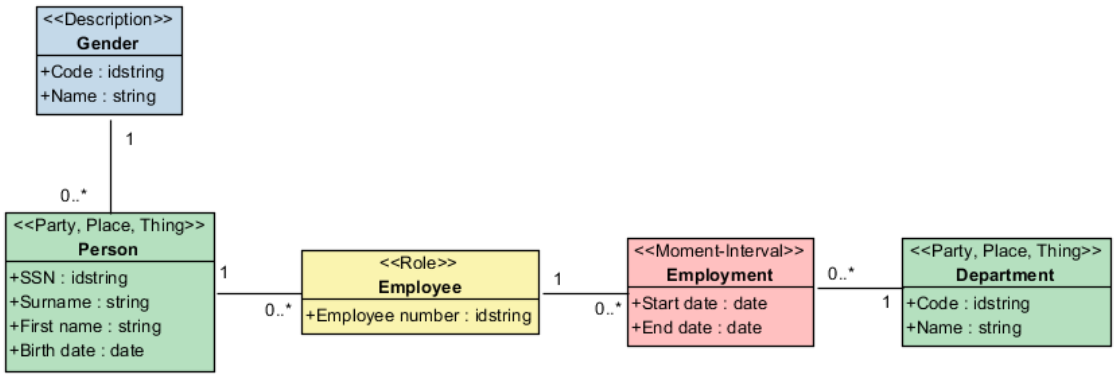
SmartEngine är ett verktyg som utvecklats som ett internt projekt. Utvecklaren, Kim Johnsson, har en bakgrund i mjukvaruutveckling. Efter att han började jobba inom informationslagerprojekt, konstaterade han att det saknades återanvändbarhet. Även om olika projekt hade liknande affärsobjekt och behov, så implementerades objekten alltid på nytt, specifikt för projektet. Han jämförde informationslagerprojekt med mjukvaruutvecklingsprojekt och ansåg att det borde gå att abstrahera skapandet av databastabellerna och laddningsprocedurerna, och därmed ha grund för att kunna automatisera genereringen och uppehållet av objekten (Johnsson, 2016).

Johnsson började sätta sig in i problemet, och år 2007 började han utveckla en lösning på problemet. På den tiden fanns det väldigt få kommersiella produkter och de som fanns var inte dugliga för produktionsbruk. Dessutom ville Johnsson ha en annorlunda synvinkel på implementation. Han ville att SmartEngine skulle vara ett verktyg för att implementera modeller. Verktyget skulle innehålla regler och logik för att läsa modeller och sedan generera slutresultatet, istället för att fungera som ett verktyg som man jobbade i. Därmed skulle det vara möjligt att utvidga egenskaperna, genom att skapa logiken för antingen utvidgade modeller, eller totalt nya modeller (Johnsson, 2016).

För tillfället kan SmartEngine generera SQL-koden som behövs för att implementera och upprätthålla ett informationslager, genom att läsa in en begreppsmodell och omvandla det till en modell av databasobjekt med förhållanden mellan sig (Johnsson, 2016).

Tidigare process för implementation av informationslager

I följande kapitel presenteras det etablerade sättet att implementera ett informationslager hos företaget, utan att gå in på detaljer. Som bas för allt arbete används en begreppsmodell. Begreppsmodellen modelleras som ett Unified Markup Language (UML) klassdiagram. Figur 13 visar en del av en modell.



Figur 13 - En del av en begreppsmodell (Skärmbild från Visual Paradigm, Eklund, 05.01.2018)

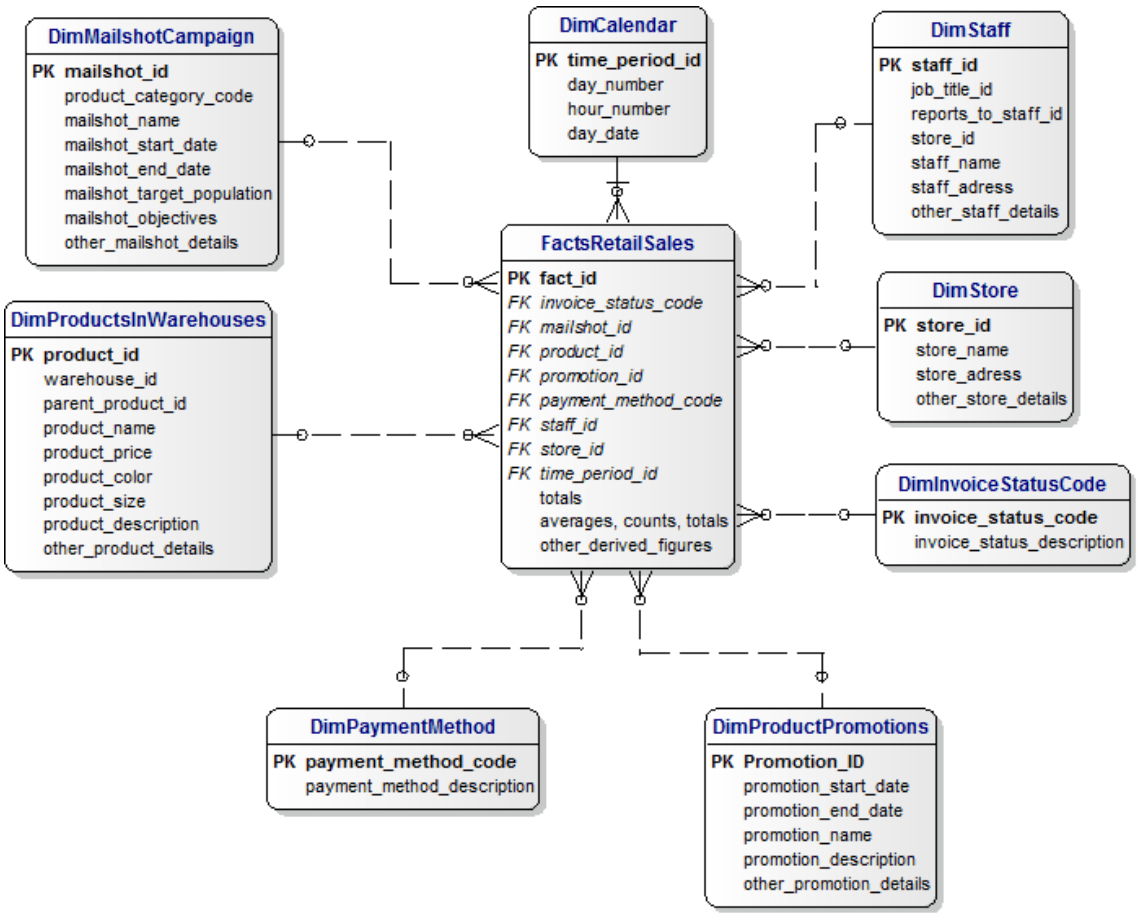
Alla affärsobjekt modelleras som entiteter med förhållanden mellan sig. Siffran betecknar kardinalitet i en viss riktning. En person har ett kön, medan ett kön innehas av många personer. En arbetstagare är alltid en person, medan en person kan under sin karriär inneha flera olika positioner, därmed vara flera olika arbetstagare.

Färgkodningen antyder typen av entiteten, närmast för tydlighet. Blå står för beskrivande entitet. Blåa är ofta kategoriseringar eller grupperingar. Gröna är verkliga, ibland fysiska objekt, som person, avdelning eller kostnadsställe. Röda är entiteter som är bundna till tidsperioder eller tidpunkter. I figur 13 har exempelvis arbetskontraktet modellerats med ett startdatum och slutdatum. Gula entiteter kan beskrivas som roller. Personen har en roll som arbetstagare. Egentligen existerar inte arbetstagare, utan de är personer, men datamässigt är rollen arbetstagare relevant.

Källdatan kan levereras eller hämtas, på några olika sätt. Det vanligaste sättet är att data exporteras till filer, som läses in i laddningsområdet i informationslagret. Ett annat vanligt sätt är att kopiera data rakt från källsystemet, via ett gränssnitt eller rakt från en databas. Filer läses oftast in med hjälp av SQL Server Integrations Services (SSIS).

SmartEngine skapar automatiskt laddningstabeller, som baserar sig på begreppsmodellen. Utvecklarens uppgift är att anpassa källdata till dessa laddningstabeller. Vanligtvis görs det genom att skapa SQL-vyer som pekar på källdata. Dessa vyer has samma struktur som tabellerna SmartEngine automatiskt skapar. Därefter utnyttjas procedurer som SmartEngine skapar, för att ladda informationslagret. I de flesta fall är databasservern Microsofts SQL Server.

Efter att informationslagret är laddat, måste utvecklaren skapa rapporteringsskiktet. Det handlar om att skapa fakta-tabeller och dimensions-tabeller, som uppfyller på förhand definierade rapporteringsbehov och -krav. Faktan skall innehålla värden och siffror, som kan aggregeras och beräknas. Dimensionerna används för att beskriva och begränsa datan. Figur 14 visar ett exempel på en fakta med många dimensioner.



Figur 14 - Exempel på en dimensionell modell (http://www.databaseanswers.org/data_models/retail_customers/pragmatic_dimensional_model.htm)

På detta sätt kan man med en och samma rapport visa företagsnivå, eller alternativt borra sig in (engelska *drill-down*) på kostnadsställen. Med en kombination av val och begränsningar stöder rapporten många användare, då alla kan tillämpa analys på den data som berör dem. Datamodellen konstrueras oftast med hjälp av SQL Server Analysis Services (SSAS). SSAS skapar datakuber menade för Online Analytical Processing (OLAP). OLAP är planerad för mångdimensionell analys av numeriska värden. I kuben definieras beräkningarna i form av mätare (engelska *measures*). Mätarna påverkas av val i de olika dimensionerna. Excel eller Power BI fungerar som de vanligaste

rapporteringsverktygen. Båda kan koppla rakt till datakuben, så värdena på rapporten är alltid de nyaste tillgängliga.

Artefakten

Målet med konstruktionen av artefakten är att behålla så mycket som möjligt av rutinerna och verktygen. Det är dock viktigt att allting skall fungera i molnet, så man inte är bunden till en viss intern dator eller server. Företaget är Microsoft partner, och valde därför Microsoft Azure som PaaS-tjänst som utnyttjas av artefakten.

Modelleringen är oförändrad, samma verktyg (Visual Paradigm) används för att rita begreppsmodellen. Källdatan kommer att lagras i Azure Storage, som är en molntjänst för datalagring. Tjänsterna i Microsoft Azure är välintegrerade, så det är ett bra alternativ att lagra data i molnet. Datan läses in till laddningsområdet av informationslagret med Azure Data Factory. Data Factory är en moln-baserad plattform menad för flyttning och transformation av data. Informationslagret byggs på Azure SQL DB, en relationsbaserad databastjänst. Azure SQL DB grundar sig på samma motor som SQL Server, och är därför en passlig ersättning för en lokal server i organisationens interna nätverk. SmartEngine måste modifieras, för att hantera vissa tekniska begränsningar som Azure SQL DB har. Ändringarna är dock små och omärkvärdiga.

Microsoft Azure erbjuder inte ännu en motsvarighet till SSAS som molntjänst. Lösningen blir att beställa en virtuell maskin i Azure-omgivningen, dit SSAS och SQL Server installeras. Lösningen uppfyller kravet om att allting skall fungera i molnet, men lösningen är inte speciellt elegant. För att Azure inte har bra stöd för integrering av Azure Active Directory (AAD) och kundens lokala Active Directory (AD), så kommer lösningen inte att uppfylla kravet, att slutanvändaren kan skriva in sig på tjänsten med sin organisations användarkonto och lösenord. Detta åtgärdas genast stöd för integrering av molnets AD och lokala AD lanseras. Excel väljs som rapporteringsverktyg, för att Excel är bekant för slutanvändarna och kundorganisation utnyttjar Excel redan från förut. Rapporten kommer att levereras åt kundens IT-avdelning, som delar rapporterna vidare internt åt slutanvändarna. Ansvar för rättigheterna till rapporterna kommer därav att ligga på kundens organisation. Då SSAS på virtuella maskinen inte kan kolla upp information om användarkontot, är det inte möjligt att implementera användarspecifika databegränsningar. Detta kan åtgärdas, då integrering av AD och AAD blir möjligt.

Projektet

Som väntat, visade sig att då möjligheten för AD-integration inte fanns, blev inloggning och datatrygghet ett problem. För att vi inte kunde begränsa data enligt vem som öppnat rapporten, blev alternativen att endast överlåta rapporterna åt några få slutanvändare, eller planera rapporterna så att ingen fick se känsliga data, eller data som kunde användas till att identifiera arbetstagare och därmed begränsa nyttan av rapporterna.

Ett tekniskt hinder i projektet var Azure Data Factorys brister. Data Factory visade sig vara ett väldigt begränsat verktyg, som inte ens klarade enkla uppgifter som att arkivera en inläst fil, utan att man själv utvecklar en metod för det med hjälp av C#. För att utvecklingsteamet inte hade erfarenhet med C#, måste problemet lösas på andra sätt, närmast via tillämpning av SQL-logik i databasen.

Mot slutet av projektet övergick man till Power BI som rapporteringsverktyg. Power BI är en molnbaserad rapporteringsomgivning, och rapporterna går att bädda in (engelska *embed*) i Sharepoint Online. På det sättet löste man problemet med att distribuera Excel-rapporterna. Det finns heller inte risk att rapporten skickas åt någon som inte borde ha tillgång till den.

Slutsats

Det visade sig att företagets arbetsprocess är passande för utnyttjande av molntjänster. För att SmartEngine är utvecklad som ett modellimplementeringsverktyg, är inte valet av databasplattform speciellt viktigt. Körningen av SQL-kod sker via andra verktyg, den bara genereras via SmartEngine. SmartEngine genererar kod enligt begreppsmodellen, som alltid har samma struktur och metadata.

En delorsak till att utnyttjandet av molntjänster gick så bra, beror på att Microsoft Azure har satsat stort på att erbjuda ersättande tjänster och verktyg för sina lokala (engelska *on-premises*) varianter. Problemen som inte kunde lösas i början av projektet, löstes till slut då AD-integration blev möjligt, då Azure Active Directory B2B lanserades. B2B möjliggör att man kan bjuda in externa användare till sin omgivning, och tilldela rättigheter och resurser som om de vore interna användare. Den andra stora nyheten var lanseringen av Azure Analysis Services (AAS), motsvarigheten till SSAS i molnet. AAS

bygger på AAD, och har därmed inbyggt stöd för radbegränsning av data, enligt användarnamnet.

Numera är informationslager i molnet ett standardval när det planeras nya lösningar åt kunder. Det finns fortfarande mycket som företaget lär sig av varje projekt, men det går tryggt att säga att informationslager i molnet är ett verkligt alternativ.

REFERENCES

Abadi, D. J. (2009). Data management in the cloud: Limitations and opportunities. *IEEE Data Eng.Bull., 32*(1), 3-12.

Agrawal, D., Das, S., & El Abbadi, A. (2011). Big data and cloud computing: Current state and future opportunities. Paper presented at the *Proceedings of the 14th International Conference on Extending Database Technology,* 530-533.

Ariyachandra, T., & Watson, H. J. (2006). Which data warehouse architecture is most successful? *Business Intelligence Journal, 11*(1), 4-6.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. (2010). A view of cloud computing. *Communications of the ACM, 53*(4), 50-58.

Assunção, M. D., Calheiros, R. N., Bianchi, S., Netto, M. A., & Buyya, R. (2015). Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing, 79*, 3-15.

Baars, H., & Kemper, H. (2010). Business intelligence in the cloud? Paper presented at the *PACIS 2010 Proceedings,* (145), 1528-1539.

Behrend, T., Wiebe, E., London, J., & Johnson, E. (2011). Cloud computing adoption and usage in community colleges. *Behaviour & Information Technology, 30*(2), 231-240.

Benbasat, I., Goldstein, D. K., & Mead, M. (1987). The case research strategy in studies of information systems. *MIS Quarterly,* 369-386.

Bojičić, I., Marjanović, Z., Turajlić, N., Petrović, M., Vučković, M., & Jovanović, V. (2016). (2016). A comparative analysis of data warehouse data models. Paper presented at the *2016 6th International Conference on Computers Communications and Control (ICCCC),* 151-159.

Breslin, M. (2004). Data warehousing battle of the giants. *Business Intelligence Journal, 7*, 6-20.

Bronson, N., Lento, T., & Wiener, J. L. (2015). (2015). Open data challenges at Facebook. Paper presented at the *2015 IEEE 31st International Conference on Data Engineering,* 1516-1519.

Calder, B., Wang, J., Ogus, A., Nilakantan, N., Skjolsvold, A., McKelvie, S., Xu, Y., Srivastav, S., Simitci, H. (2011). (2011). Windows azure storage: A highly available cloud storage service with strong consistency. Paper presented at the *SOSP '11 Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles,* 143-157.

Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP technology. *ACM Sigmod Record, 26*(1), 65-74.

ComPUtING, C. (2011). Cloud computing privacy concerns on our doorstep. *Communications of the ACM, 54*(1), 36-38.

Conn, S. S. (2005). (2005). OLTP and OLAP data integration: A review of feasible implementation methods and architectures for real time data analysis. Paper presented at the *SoutheastCon, 2005. Proceedings. IEEE,* 515-520.

Cuzzocrea, A., Bellatreche, L., & Song, I. (2013). (2013). Data warehousing and OLAP over big data: Current challenges and future research directions. Paper presented at the *Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP,* 67-70.

Davis, F. D. (1985). *A technology acceptance model for empirically testing new end-user information systems: Theory and results* (Doctoral dissertation). Retrieved from https://dspace.mit.edu/bitstream/handle/1721.1/15192/14927137-MIT.pdf?sequence=2

Devlin, B. A., & Murphy, P. T. (1988). An architecture for a business and information system. *IBM Systems Journal, 27*(1), 60-80.

Dhont, J., Asinari, M. V. P., Poullet, Y., Reidenberg, J. R., & Bygrave, L. A. (2004). Safe harbour decision implementation study. *Universität Namur, Bericht Für Die Europäische Kommission, Generaldirektion Binnenmarkt Und Dienstleistungen,*

Eckerson, W. (2015a). *Attunity product profile report*. Eckerson.com: Eckerson Group. Retrieved from http://eckerson.com/articles/which-data-warehouse-automation-tool-is-right-for-you

Eckerson, W. (2015b). *Data warehouse automation tools*. Eckerson.com: Eckerson Group. Retrieved from http://eckerson.com/articles/which-data-warehouse-automation-tool-is-right-for-you

Eckerson, W. (2015c). *Magnitude product profile report*. Eckerson.com: Eckerson Group. Retrieved from http://eckerson.com/articles/which-data-warehouse-automation-tool-is-right-for-you

Eckerson, W. (2015d). *TimeXtender product profile report*. Eckerson.com: Eckerson Group. Retrieved from http://eckerson.com/articles/which-data-warehouse-automation-tool-is-right-for-you

Eckerson, W. (2015e). *WhereScape product profile report*. Eckerson.com: Eckerson Group. Retrieved from http://eckerson.com/articles/which-data-warehouse-automation-tool-is-right-for-you

Fuggetta, A., & Di Nitto, E. (2014). Software process. Paper presented at the *Proceedings of the on Future of Software Engineering,* 1-12.

George, S. (2012). Inmon vs. Kimball: Which approach is suitable for your data warehouse? Retrieved from http://www.computerweekly.com/tip/Inmon-vs-Kimball-Which-approach-is-suitable-for-your-data-warehouse

Goldberg, K. (2012). What is automation? *IEEE Transactions on Automation Science and Engineering, 9*(1), 1-2.

Golfarelli, M., & Rizzi, S. (2009). *Data warehouse design: Modern principles and methodologies*, McGraw-Hill, Inc.

Grant, T. (2015). New buzzword! data warehouse automation. Retrieved from https://www.jetreports.com/2015/03/05/new-buzzword-data-warehouse-automation/

Iivari, J. (2003). The IS core - VII: Towards information systems as a science of meta-artifacts. *Communications of the Association for Information Systems, 12*(37), 568-581.

Inmon, W. H. (2005). *Building the data warehouse*, John Wiley & Sons.

Inmon, W. H., Imhoff, C., & Sousa, R. (2002). *Corporate information factory*, John Wiley & Sons.

Johnsson, K. (2016). *Interview about SmartEngine, personal interview, 2016.11.03*

Jovanovic, V., & Bojicic, I. (2012). Conceptual data vault model. Paper presented at the *Proceedings of the Southern Association for Information Systems Conference,* Atlanta, GA, USA.*, 23*, 131-136.

Jukic, N. (2006). Modeling strategies and alternatives for data warehousing projects. *Communications of the ACM, 49*(4), 83-88.

Kakish, K., & Kraft, T. A. (2012). ETL evolution for real-time data warehousing. Paper presented at the *Proceedings of the Conference on Information Systems Applied Research ISSN, 2167,* 1508.

Kasanen, E., Lukka, K., & Siitonen, A. (1993). The constructive approach in management accounting research. *Journal of Management Accounting Research, 5*, 243.

Kimball, R., & Ross, M. (2013). *The data warehouse toolkit: The definitive guide to dimensional modelling*, John Wiley & Sons.

Kimball, R., Ross, M., & Thornthwaite, W. (2011). *The data warehouse lifecycle toolkit (2)*. Hoboken, US: Wiley.

Kimpel, J. F., & Morris, R. (2013). Critical success factors for data warehousing: A classic answer to a modern question. *Issues in Information Systems, 14*(1), 376-384.

Krishnan, K. (2013). *Data warehousing in the age of big data*, Morgan Kaufmann.

Lee, Y., Kozar, K. A., & Larsen, K. R. T. (2003). The technology acceptance model: Past, present, and future. *Communications of the Association for Information Systems, 12*(50), 752-780.

Lee, J. D., & See, K. A. (2004). Trust in automation: Designing for appropriate reliance. *Human Factors, 46*(1), 50-80.

Linstedt, D., & Olschimke, M. (2015). *Building a scalable data warehouse with data vault 2.0*, Morgan Kaufmann.

Luhn, H. P. (1958). A business intelligence system. *IBM Journal of Research and Development, 2*(4), 314-319.

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., & Ghalsasi, A. (2011). Cloud computing—The business perspective. *Decision Support Systems, 51*(1), 176-189.

Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing.* (Special Publication No. 800-145) National Institute of Standards and Technology. Retrieved from http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

Miah, M. B. A., Hasan, M., & Uddin, M. K. (2015). A new HDFS structure model to evaluate the performance of word count application on different file size. *International Journal of Computer Applications, 111*(3)

Microsoft. (2017a). Comparing tabular and multidimensional solutions. Retrieved from https://docs.microsoft.com/en-us/sql/analysis-services/comparing-tabular-and-multidimensional-solutions-ssas

Microsoft. (2017b). SQL server management studio (SSMS). Retrieved from https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms

Microsoft. (2017c). What is analysis services? Retrieved from https://docs.microsoft.com/en-us/sql/analysis-services/analysis-services

Microsoft. (2017d). What is the azure SQL database service? Retrieved from https://docs.microsoft.com/en-us/azure/sql-database/sql-database-technical-overview

Microsoft. (2018a). Introduction to azure data factory. Retrieved from https://docs.microsoft.com/en-us/azure/data-factory/v1/data-factory-introduction

Microsoft. (2018b). Monitor and manage azure data factory pipelines by using the monitoring and management app. Retrieved from https://docs.microsoft.com/en-us/azure/data-factory/v1/data-factory-monitor-manage-app

Mundy, J., & Thornthwaite, W. (2007). *The Microsoft data warehouse toolkit: With SQL server 2005 and the Microsoft business intelligence toolset*, John Wiley & Sons.

Parasuraman, R., Sheridan, T. B., & Wickens, C. D. (2000). A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 30*(3), 286-297.

Pemberton, J. D., & Robson, A. J. (2000). Spreadsheets in business. *Industrial Management & Data Systems, 100*(8), 379-388.

Rossi, B. (2015). The data warehouse isn't dead: It just needs an automation overhaul. Retrieved from http://www.information-age.com/data-warehouse-isnt-dead-it-just-needs-automation-overhaul-123459596/

Russo, M. (2014). *SSAS tabular as analytical engine.* Unpublished manuscript. Retrieved 2018/01/23, Retrieved from https://www.sqlbi.com/wp-content/uploads/Tabular-as-Analytical-Engine.pdf

Sarter, N. B., Woods, D. D., & Billings, C. E. (1997). Automation surprises. *Handbook of Human Factors and Ergonomics, 2*, 1926-1943.

Sen, A., Ramamurthy, K., & Sinha, A. P. (2012). A model of data warehousing process maturity. *IEEE Transactions on Software Engineering, 38*(2), 336-353.

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop distributed file system. Paper presented at the *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST),* 1-10.

Šubić, T., Poščić, P., & Jakšić, D. (2015). Big data in data warehouses. Paper presented at the *5th International Conference the Future of Information Sciences (INFuture'15),* 235-244.

Svantesson, D., & Clarke, R. (2010). Privacy and consumer risks in cloud computing. *Computer Law & Security Review, 26*(4), 391-397.

Tan, X., & Kim, Y. (2015). User acceptance of SaaS-based collaboration tools: A case of google docs. *Journal of Enterprise Information Management, 28*(3), 423-442.

TDWI. (2015). *TDWI E-book | data warehouse automation: Accelerating business results.* (E-book). tdwi.org: TDWI. Retrieved from https://tdwi.org/research/2015/11/ebook-dw-automation/asset.aspx?tc=assetpg&tc=page0

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Antony, S., Liu, H., Murthy, R. (2010). Hive-a petabyte scale data warehouse using Hadoop. Paper presented at the *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010),* 996-1005.

Turban, E., Rainer, R. K., & Potter, R. E. (2007). *Introduction to information systems: Supporting and transforming business*, John Wiley & Sons, Inc.

Watson, H. J., Goodhue, D. L., & Wixom, B. H. (2002). The benefits of data warehousing: Why some organizations realize exceptional payoffs. *Information & Management, 39*(6), 491-502.

Williams, C. N. (2015). *Testing data vault-based data warehouse* (Master's Thesis). (1340). Retrieved from https://digitalcommons.georgiasouthern.edu/etd/1340

Wu, W., Lan, L. W., & Lee, Y. (2013). Factors hindering acceptance of using cloud services in university: A case study. *The Electronic Library, 31*(1), 84-98.

Yin, R. K. (1981). The case study crisis: Some answers. *Administrative Science Quarterly, 26*(1), 58-65.